A PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

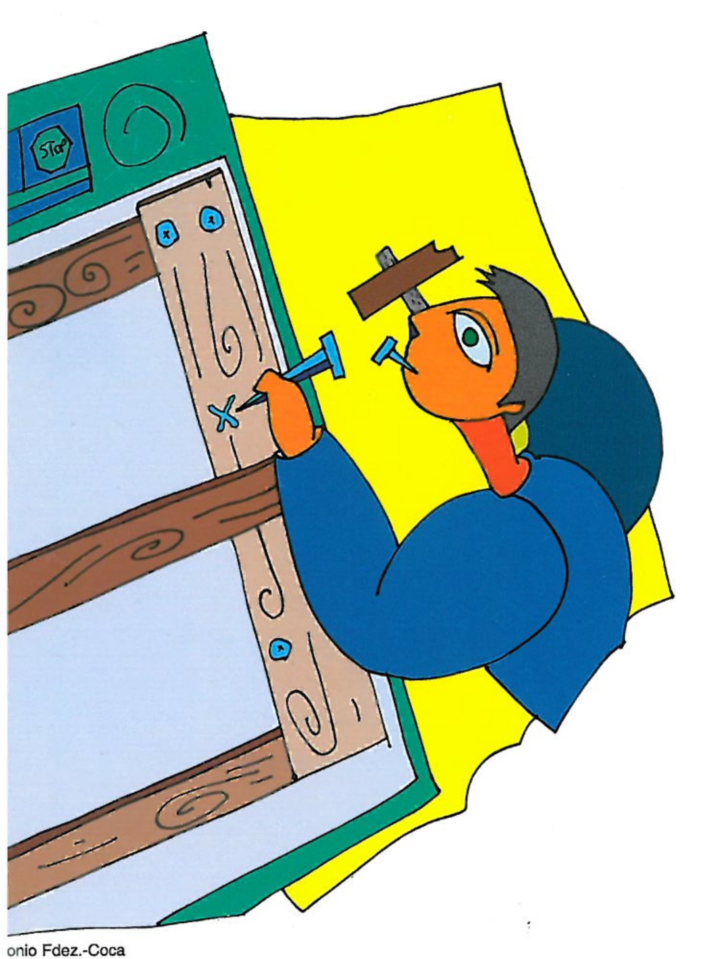
PROGRAMADORES

V, NÚMERO 51

975 Ptas.

HTML DINAMICO COMPATIBLE

GESTION DE LAS CAPAS Y EJEMPLOS PRACTICOS



onio Fdez.-Coca

CONTENIDO DEL CD-ROM

- DRUMBEAT 2.01
- NETSCAPE VISUAL JAVASCRIPT 1.0
- DYNAMIC HTML EDITING COMPONENT SDK
- HOMESITE 4.0 BETA RC-3

MULTIMEDIA
Reconocimiento de voz

REDES LOCALES
Comportamiento de los protocolos

SEGURIDAD
Herramientas para construir un cortafuegos

HERRAMIENTAS
Delphi 4: ¿alguien da más?

JAVA
Introducción a los JavaBeans

DESARROLLO EN CORBA
Programación distribuida orientada a objetos con CORBA

TOWER

8 413042 303299

PROGRAMADORES

Número 51
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS
Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo De Riquer Frutos
eriquer@towercom.es
Redactora Jefe
Cristina Peña del Pozo
crisp@towercom.es
Colaboradores
Constantino Sánchez, Jorge Figueroa, Juan J. Taboada, José J. Mora, Jordi Agost, Javier Sanz, Adolfo Aladro, Enrique de la Lastra, Enrique Díaz, J.L. Alvarez, Antonio Ruiz.
Maquetación
Javier Pérez
Tratamiento de Imagen
Clara Francés

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Alicia Zazo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
oscarrf@towercom.es
Javier Amado
jamado@towercom.es
Servicio Técnico
Manuel Hernando
mhernando@towercom.es

Preimpresión
Indes Color
Impresión
Gráficas Muriel
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN



Menos información, más conocimiento

Como suarios de la Informática estamos de enhorabuena otra vez, pues comienza una nueva época que a buen seguro colmará todas nuestras aspiraciones (o no, claro). En principio desconozco cuántas de estas etapas han surgido en estos últimos años, ni cuántas han cumplido realmente sus expectativas, pero de lo que no hay duda es de que se convertirá en otra de las múltiples zanahorias que de vez en cuando aparecen ante nuestros ojos.

Si bien se ha producido una alternancia entre éxitos y fracasos, para ser justos debemos reconocer que esta forma de actuar (que por otro lado es la que hay...) confiere dinamismo e ilusión a todos los involucrados.

Por cierto, pido perdón por no haber presentado a tan insigne actor que aparece en el panorama actual, conocido por los entendidos como Gestión del Conocimiento, cuyo futuro a nivel empresarial resulta en principio más que prometedor. Su objetivo principal pasa por la obtención, clasificación y administración de grandes volúmenes de información , con el fin de ofrecer los datos exactos y requeridos en el momento oportuno, con la mayor brevedad posible. La palabra clave en este caso es requeridos, porque no se pretende dar ni más ni menos información que la precisa y adecuada al problema actual.

De la sociedad de la información se pretende saltar a un concepto superior donde no predomine la cantidad sino la calidad (y la cantidad), pero en este orden. El acceso, gestión, actualización y presentación de los datos requeridos de forma precisa y eficaz se convertirá en otro de los objetivos de la programación, ya que se requerirán herramientas muy específicas para la toma de decisiones. Desde luego la labor más complicada consiste en aunar el conocimiento procedente de varias fuentes y clasificarlo en funciones de unos patrones de similitud que relacionen los conceptos, ya que en multitud de ocasiones la relación no resulta nada evidente.

En realidad este término tan atractivo y actual se apoya en una idea tan evidente y lógica que parece mentira que se "descubra" ahora, y que se ha reflejado en la red Internet. En sus desapercibidos comienzos, cuando parecía que no había nada que contar, se encontraba lo que se necesitaba con mayor rapidez que en la actualidad debido a que ante una consulta se reciben demasiados datos, sin conocer a priori cuáles tendrán relevancia. Al principio los buscadores se convirtieron en la solución universal, incorporando con el tiempo filtrados, refinamientos, acotaciones del dominio, clasificaciones, etc. Pero también se ha demostrado que adolecen de ciertas características inteligentes que les permitan separar el enorme volumen de datos que manejan.

En fin, la solución del caso no la conoceremos hasta dentro de no mucho tiempo (menos del que pensamos casi seguro), pero antes de que la lleguemos siquiera a intuir, tendremos una nueva zanahoria a nuestro alcance que nos hará olvidar si conseguimos comernos la anterior.

SÓLO PROGRAMADORES



16 Seguridad
HERRAMIENTAS PARA CONSTRUIR UN CORTAFUEGOS (II)

Si el mes pasado nos detuvimos para conocer los co ceptos más básicos, ahora es el momento adecua para conocer algunas de las soluciones.

22 Programación Multimedia
RECONOCIMIENTO DE VOZ (IV)

Enlazando con el capítulo anterior continuamos ad lante, y en esta ocasión trataremos de aprender a ac var el programa encargado de ajustar correctamente micrófono.

36 WWW
LAS CAPAS EN HTML DINÁMICO (I)

Las innovaciones nunca se detienen en el campo del software, y en el artículo de portada vamos a conocer algunas de ellas. Dedicamos estas páginas al DHTML, una de las novedades más importantes para todos aquellos que estén dispuestos a manipular los elementos de una página web. Una tecnología con sus ventajas e inconvenientes que vamos a ir desgranando.

6 Noticias
NOVEDADES

Estas páginas os van a poner al corriente de todo aquello que sucede y que os interesa conocer en el mercado de la tecnología del campo informático. Todo lo que está ocurriendo en este entorno a través de nuestras páginas.

10 C/C++
PROGRAMACIÓN BÁSICA EN WINDOWS (y V)

Damos por finalizada esta serie de artículos, no sin antes detenernos en la utilización de los famosos menús de opciones y las barras de herramientas, unos de los recursos más usados en el desarrollo de las aplicaciones actuales.

29
Java
INTRODUCCIÓN A LOS JAVABEANS

¿Qué es un JavaBean?, ¿para qué se utiliza?; éstas son algunas de las preguntas a las que daremos respuesta a lo largo de este espléndido artículo que de nuevo nos llevará hasta uno de los conceptos claves del mundo Java.

55
Visual Basic
PROGRAMACIÓN DEL API DE WINDOWS (y IV)

Terminamos esta serie con unos ejemplos prácticos que nos permitirán aplicar toda la teoría que hemos aprendido en los artículos anteriores. Veremos cómo podemos conectarnos o desconectarnos de unidades de discos a través de una red o cómo lograr que un determinado formulario aparezca siempre visible por encima de cualquier otro.

60
Redes Locales
INTERCONEXIÓN DE REDES (I)

Si en artículos anteriores tocamos el tema del cableado de red, ahora es el momento de detenernos y adentrarnos en el funcionamiento de los protocolos viendo los diferentes tipos como los de contienda Polling y Token.

68
Programación CORBA,
PROGRAMACIÓN DISTRIBUIDA ORIENTADA A OBJETOS CON CORBA (I)

En esta primera parte haremos una toma de contacto con los aspectos más importantes e interesantes de la programación distribuida orientada a objetos basada en la tecnología CORBA.

74
Libros
ACTUALIDAD

En nuestra sección dedicada a los libros os presentamos este mes cuatro ejemplares de gran interés para todos los gustos, para aquellos que estéis interesados en el desarrollo de los JavaBeans, los que programéis en Delphi, para aquellos que trabajen con bases de datos en SQL, o para los que utilizan HTML Dinámico en las composiciones de sus web.

47 Herramientas de desarrollo DELPHI 4: ¿ALGUIEN DA MÁS?

Una nueva versión de esta herramienta ha hecho su aparición en el mercado. Es el momento de pararnos a examinarla y ver las novedades que puede aportar a todos los que trabajan en el ámbito de Delphi: mejoras en la interfaz, creación asistida del código, soporte de CORBA, un lenguaje más potente drivers nativos de Access 97, etc. Todo lo que aporta Delphi 4 está aquí.

78
CONTENIDO DEL CD-ROM

Como siempre un nuevo CD-ROM que se encarga de complementar nuestra revista. Una vez más nos hemos encargado de buscar lo último en programación para que podáis estar siempre al día de las ofertas que os presenta este entorno. Este mes os proporcionamos programas tan atractivos como: Drumbeat 2.01, Netscape Visual JavaScript 1.0, WinZip 7.0 o UnixDos Toolkit 4.2,etc.



Netscape refuerza su estrategia hacia Linux mediante su entrada en el capital de Red Hat Software

TODOS SE SUBEN AL CARRO DEL SISTEMA LINUX

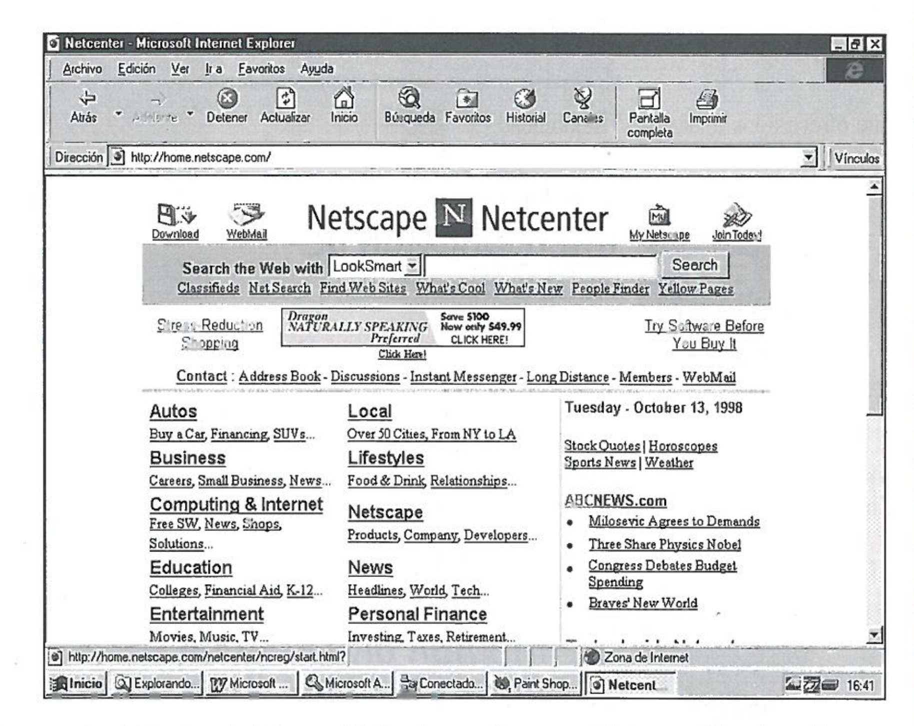
Netscape Communications Corporation ha decidido apostar muy fuerte por Linux, para lo que ha tomado posiciones minoritarias, junto con Intel Greylock y Benchmark Partners, en Red Hat Software, el principal distribuidor de este popular sistema operativo de código fuente abierto, que posee una importante cuota del mercado para empresas y proveedores de servicios de Internet.

Al mismo tiempo, la compañía ha anunciado sus planes para soportar Linux como plataforma estratégica en el lado servidor.

Actualmente Netscape ya soporta este sistema en el lado cliente. De hecho, la versión beta del Communicator 4.5 para Linux es la más descargada, desde el Netcenter, de todas la plataformas Unix que soporta Netscape.

Red Hat Software desarrolla, mantiene y proporciona soporte técnico para el sistema operativo, además ha comentado la creación de la nueva división Enterprise computing Division que ofrecerá servicios y productos empresariales para soportar aplicaciones críticas de negocio.

Según International Data Corporation, el software servidor que más ha crecido entre 1996 y 1997 fue Linux. Este crecimiento le posiciona en el quinto lugar en este segmento de mercado de software, además Linux



es también la plataforma Unix de mayor crecimiento en la empresa con una estimación de 7 millones de usuarios, según Linux Online.

El software servidor de Netsccape para Linux ofrecerá a los clientes empresariales de gama alta la disponibilidad y escalabilidad del software servidor de Netscape, y la innovación y el soporte de estándares abiertos de Linux.

En concreto Netscape tiene previsto ofrecer proximamente las versiones en Linux de su software servidor, empezando por Netscape Messaging Server y Netscape Directory Server. Estos productos estarán disponibles en el mercado para todos quellos interesados durante el primer trimestre de 1999.

Las direcciones de interés son: www.linux.org, www.redhat.com/ y http://home.netscape.com/es

Si desea obtener la información o la licencia del software Java Shared Data Toolkit puede consultar en la siguiente dirección web:

http://java.sun.com/products/java-media/jsdt

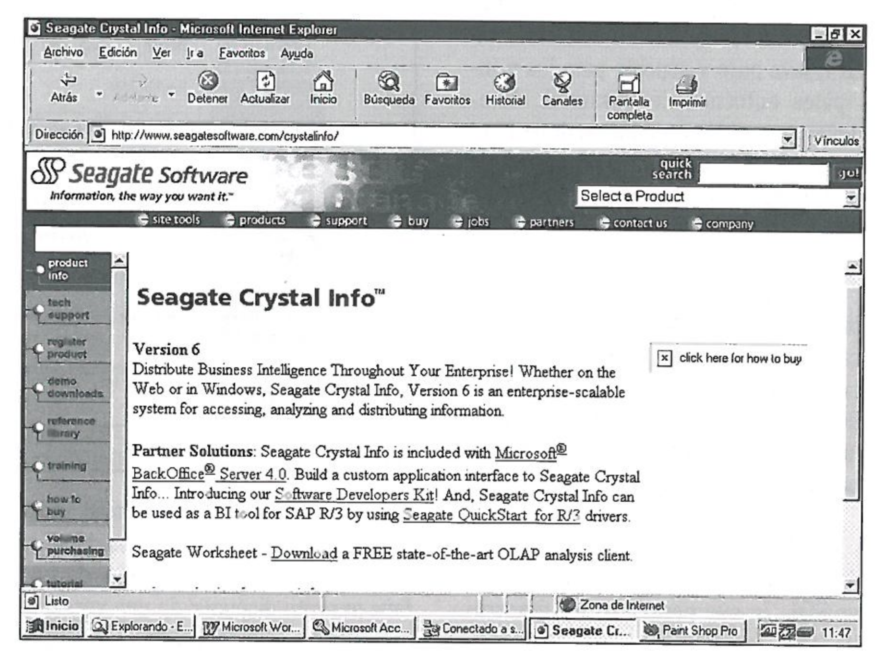
SEGATE SOFTWARE E INFORMIX AUNAN FUERZAS PARA OFRECER POTENTES Y ESCALABLES SOLUCIONES DATA WAREHOUSING

Seagate Software, suministrador de software de Business intelligence, ha anunciado un acuerdo con Informix para incluir Seagate Crystal Info 6.0 en Decision Frontier Solution Suite de Informix.

Seagate Crystal Info 6.0 es un sistema de análisis y creación de informes que ofrece a los usuarios la flexibilidad para formatear y distribuir información crítica corporativa, ofreciendo un acceso nativo a Decision Frontier Solution Suite, que integra Informix Dynamic Server con Advanced Decision Support Option e Informix MetaCube OLAP Option

Seagate Crystal Info 6.0 es una herramienta que se encarga de aprovecha las capacidades de la tecnología Channel Push como un camino supletorio para recibir un rápido acceso a la información a través de la web.

La tecnología Channel Push permite a los usuarios transmitir informes MetaCube, de forma que estén informados de cualquier actualización que haya tenido lugar.



Los usuarios de este nuevo producto pueden asegurar, visualizar y analizar la creación de informes compartida desde su explorador web, utilizando Java, HTML, o ActiveX para una completa flexibilidad en su entorno preferido.

Cinco licencias cliente de Seagate Crysta Info 6.0 y una licencia Report/Query estará disponible gratutamente junto con MetaCube 4.0 en el tercer trimestre del presente año.

Las direcciones en las que puede ampliar esta información en caso de estar interesado en el tema son:

www.seagatesoftware.com y www.informix.com

PLATINUM TECHNOLOGY SOPORTA MICROSOFT VISUAL STUDIO 6.0 PARA EL DESARROLLO DE APLICACIONES CORPORATIVAS

A partir de ahora todos aquellos desarrolladores que utilizan habitualmente Visual Studio ya pueden emplear las herramientas de PLATINUM desde este mismo momento, para poder realizar tareas tales como la modelización de empresa, desarrollo de código de bases de datos y reglas de negocio, gestión de cambios y configuración, y desarrollo y presentación de gráficos 3D interactivos incluidos en las aplicaciones, entre otras muchas diferentes opciones.

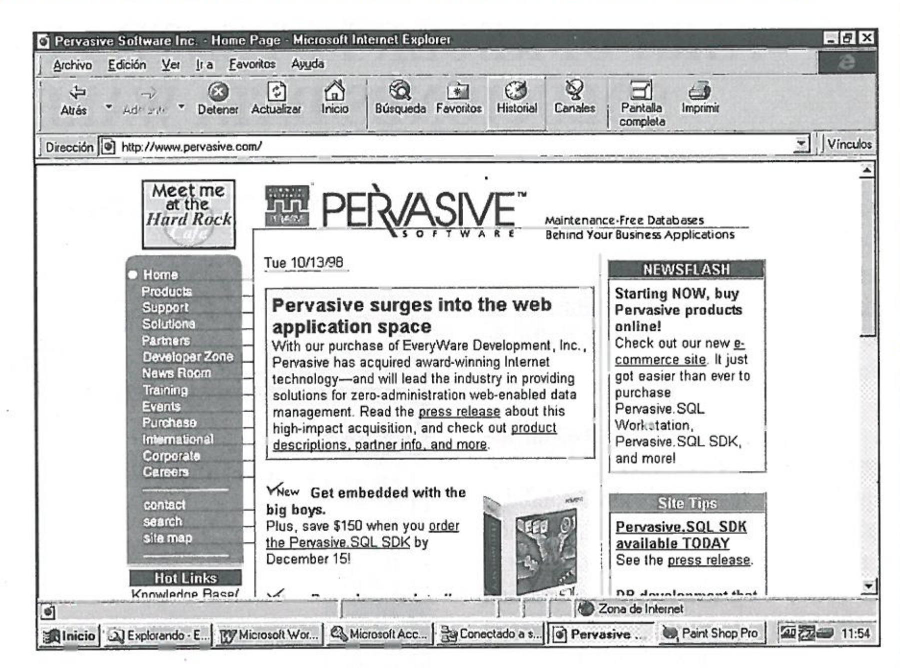
La combinación de Microsoft con lo productos de PLATINUM crea una completa y potente solución para desarrollar rápidamente y gestionar aplicaciones utilizando Microsoft Visual Basic, Visual J++ y Visual C++.

KIT PARA DESARROLLADORES DE PERVASIVE

Pervasive ha presentado el nuevo kit para desarrolladores Pervasive.SQL Software Developers Kit (SDK). Este completo conjunto de recursos de desarrollo de aplicaciones incluye I*net Data Server, controles ActiveX, un API, Java puro y soporte para los principales entornos de desarrollo Windows para acelerar de forma drástica el desarrollo de aplicaciones basadas en el motor de bases de datos embebido ultraligero de Pervasive.

La utilidad I*net Data Server permite a los desarrolladores escribir aplicaciones basadas en Pervasive.SQL en cualquiera de los entornos de desarrollo Windows más populares y ejecutarlas como soluciones cliente/servidor basadas en la Web o Internet.

El nuevo SDK incorpora el acceso ActiveX nativo al interfax de Pervasive.SQL, lo que permite a los desarrolladores escribir directamente en la base de datos desde los conocidos entornos Delphi y Visual Basic. Además de todo esto, al incluir también un interfaz nativo Java y librerías de tipo Java permite disponer de una verdadera programación orientada a objeto con Pervasive.SQL



El SDK, que ya está disponible, se comercializará hasta finales de año a un precio promocional de $149, pasado este periodo el precio será de $295. La dirección en la que lo puede adquirir es www.pervasive.com

MICROSOFT INVIERTE EN LA FORMACIÓN DE LOS DESARROLLADORES

Microsoft ha anunciado una inversión de 450 millones de pesetas en el desarrollo de tres programas de formación los más de 200.000 profesionales del sector de las Tecnologías de la Información.

Este anuncio, enmarcado dentro de la iniciativa más estratégica de Microsoft, en la que invertirá treinta y cinco mil millones de pesetas y que irá dirigida a 20 millones de profesionales de todo el mundo.

La iniciativa incluye tres programas para tres grupos distintos: programa Microsoft TechNet para ingenieros y técnicos en informática, MSDN para desarrolladores de software y por último Microsoft Acceso Directo para distribuidores de informática.

El MSDN es la iniciativa con la que Microsoft tse ha marcado como principal objetivo satisfacer las demandas y ofrecer recursos a las comunidades técnicas y de desarrolladores. Creada en 1994, ofrece mediante una suscripción, toda la información acerca de productos, tecnologías, foros de debate, etc., ahora se completa con la organización de numerosos eventos clasificados en tres tipos.

MSDN Presentaciones es el programa, en el que se van a tratar diversos temas de distinta índole tales como el diseño y desarrollo de una aplicación de negocio.

MSDN Developer Day, en este evento se podrá conocer, de la mano de Bill Gates, el concepto de Sistema Nervioso Digital y Visual Studio 6.0.

El tercero y último es MSDN Foro de desarrolladores con contenidos de carácter técnico altamente prácticos para cubrir todas las necesidades de un desarrollador.

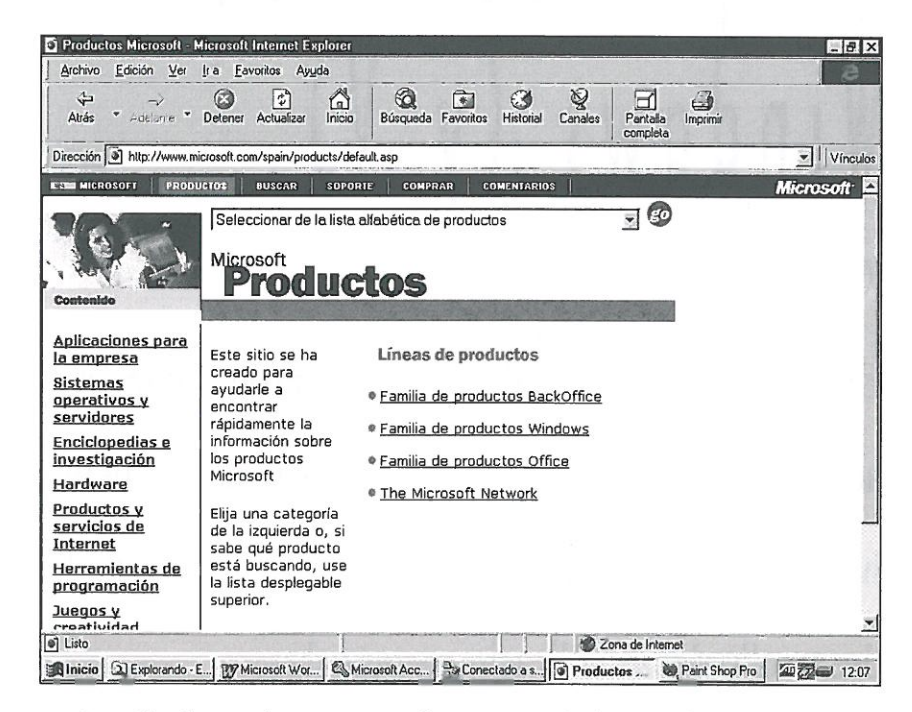
Si desea inscribirse puede hacerlo a través de Internet en la siguiente dirección: www.microsoft.com/spain/msdn/

MICROSOFT ANUNCIA SU ESTRATEGIA DE ANÁLISIS DE DATOS CON MICROSOFT SQL SERVER 7.0 Y MICROSOFT OFFICE 2000

Microsoft anuncia su estrategia de análisis de datos con Microsoft SQL Server 7.0 y Microsoft Office 2000.

El gigante del software anuncia la interoperabilidad existente entre Microsoft Office 2000 y la versión recientemente anunciada Microsoft SQL Server 7.0, la base de datos más popular para el sistema operativo Windows NT.

La nueva versión que ahora aparece de Office 2000 incluirá nuevas capacidades de Excel 2000, Access 2000, de esta forma los usuarios de Excel pueden construir aplicaciones Access totalmente compatibles con Microsoft SQL Server, lo que nos permitirá visualizar y analizar datos multidimensionales y relacionales en data warehouses y data marts basados en SQL Server 7.0.



Esta integración entre las nuevas versiones representa un paso en las alianzas de la compañía Microsoft que permitirá ofrecer soluciones como data warehousing, comercio electrónico y aplicaciones de negocio que se integran con las aplicaciones sobremesa que utilizan los clientes.

Sql Server 7.0 es la base de datos compatible con el sistema operativo Windows NT, que ofrece grandes e importantes ventajas corporativas y mejoras en la toma de decisiones a todos los niveles de la organización, a través de la potencia de las soluciones industriales y el data warehousing y la interoperabilidad con Microsoft Office 2000.

Si usted desea ampliar la información respecto a esta noticia, la puede obtener en Internet, en la siguiente dirección:www.microsoft.com/spain/

CONFERENCIA PARA DESARROLLADORES

Si le interesa conocer lo último en tecnologías de software para desarrolladores, la compañía Sun le ofrece la posibilidad ahora de hacerlo en un solo día gracias a una conferencia gratuita llevada a cabo por sus expertos y jefes de producto.

El día 11 de noviembre en Barcelona podrá informarse sobre el nuevo programa Sun Developer Connection, conocer lo último en herramientas, productos y ofertas tecnológicas de esta compañía.

Las inscripciones para todos los que quieran asistir son gratuitas vía web en: www.sun.es/desarrolladores o en www.sun.com/developers/ europeshow.



SÓLO PROGRAMADORES

Programación básica en Windows (y V)

Jorge Figueroa (erde@arrakis.es)

En este último capítulo sobre programación básica bajo Windows nos adentraremos en la utilización de los famosos menús de opciones y las barras de herramientas. Sin duda alguna, se trata de dos de los recursos más usados en el desarrollo de las aplicaciones actuales.

En el artículo del mes pasado se explicó la forma de poder aprovechar los recursos más sencillos en nuestros propios programas tales como gráficos, iconos, cursores y sonidos. En este punto explicaremos el funcionamiento de los menús de opciones y las barras de a herramientas, ya que se trata de dos recursos de Windows que se emplean en la práctica totalidad de las aplicaciones.

LOS MENÚS Y LAS BARRAS DE HERRAMIENTAS

Los menús, como casi todos los lectores sabrán ya, son un recurso que permiten al usuario poder apuntar y seleccionar con el ratón una opción determinado de un amplio conjunto. Realizan su función de forma rápida e intuitiva, ofreciendo una respuesta adecuada en función de la elección del usuario. Por otra parte, las barras de herramientas proporcionan una forma de acceder a las opciones de menú más usadas de forma rápida (haciendo un solo clic de ratón sobre el icono correspondiente) sin necesidad de navegar por los menús y tener que buscar la opción deseada.

LOS MENÚS

Las opciones que incluye un menú, suelen ser simples elecciones, aunque también puede incluir cuadros de diálogo y gráficos. Como se indicó anteriormente, un menú es un recurso, y por lo tanto se debe definir en los llamados ficheros de recursos con extensión **rc**, que una vez compilados, dan lugar a un fichero de tipo **res.** A continuación se muestra un ejemplo de una definición de un menú:

```
#include "windows.h"
#include "menutool.h"

IDTB_BITMAP BITMAP "menutool.bmp"

Alteration MENU
 {
  POPUP "&Ellipse_size"
   {
    MENUITEM "&Small",   ID_SMALL
    MENUITEM "&Medium", IDM_MEDIUM
    MENUITEM "&Large",   IDM_LARGE
   }
  POPUP "&Background_Color"
   {
    MENUITEM "&RED\tF1",     IDM_RED
    MENUITEM "&GREEN\tF2", IDM_GREEN
    MENUITEM "&BLUE\tF3",    IDM_BLUE
   }
```

```
}
Alteration ACCELERATORS
 {
  VK_F1, IDM_RED,     VIRTKEY
  VK_F2, IDM_GREEN,   VIRTKEY
  VK_F3, IDM_BLUE,    VIRTKEY
 }
```

Como descubrirá el lector con la práctica, un menú puede usar muchas palabras claves tales como *MENU*, *POPUP*, *MENUITEM*, *MENUBAR-BREAK*, y que ayudan en conjunto, a poder definir de forma sencilla y fácil los menús de opciones. Algunos de estos atributos se detallan más adelante a lo largo del artículo.

Las barras de herramientas son una novedad que apareció con Windows 95

El nombre con el que se define el menú puede ser tanto un nombre real como un número identificativo. En el primer caso se puede hacer que se muestre con el carácter que queramos subrayado (colocando el carácter & delante de la letra deseada), lo que permite que después se pueda seleccionar el menú pulsando la combinación rápida de teclas: *alt + tecla subrayada*. En el segundo el menú tendrá que estar definido en un fichero de cabecera (.h) para que éste pueda ser usado en cualquier programa.

Cuando se define más de un menú conviene recordar señalar que éstos se añaden a la barra de menús de izquierda a derecha, por lo que cada menú que añadamos a la barra aparecerá situado en pantalla (en la ventana) a la derecha del anterior que haya sido definido.

OPCIONES DEL MENÚ

En el aspecto referente a las propias opciones definidas para un menú, se debe tener en cuenta la posibilidad de emplear la técnica de selección rápida comentada en el punto anterior. La única restricción que debemos tener en cuenta es que no está permitido que dos opciones de un mismo menú usen la misma letra de acceso rápido por teclado.

Las opciones de menú pueden ser de muchas clases, y por lo tanto disponemos de muchos atributos para poder definirlas. A continuación se detallan los más utilizados:

- *CHECKED*: El elemento de menú tiene una marca de verificación a su izquierda indicando que se encuentra seleccionado.
- *END*: El elemento del menú es el último elemento de un menú desplegable o estático.
- *GRAYED*: En este caso el elemento de menú no está activado, lo que implicará que se visualizará en pantalla con su texto de opción en color gris oscuro.
- *INACTIVE*: Este atributo hace que la opción se muestre pero que no pueda ser seleccionada para su uso por parte del usuario.
- *MENUBREAK*: El elemento de menú se sitúa en una columna nueva.
- *MENUBARBREAK*: El elemento de menú se sitúa en una nueva columna separada de las anteriores por una barra separadora.
- *OWNERDRAW*: El propio menú es responsable de dibujar todos los elementos visuales del menú, incluyendo los modos destacados, inactivos y verificados en que pueden encontrarse las opciones.
- *POPUP*: Cuando esta opción está seleccionada hace que se muestre una lista de elementos.

LAS BARRAS DE HERRAMIENTAS

Junto con la aparición de Windows 95 y NT, se hicieron rápidamente populares una nueva clase de recurso, las barras de herramientas. Para el programador, la nueva función *CreateToolbar Ex()* permite poder añadir fácilmente una barra de herramientas en un programa. A continuación se muestra un ejemplo de barra donde aparecen tres gráficos formados por unos pequeños iconos. Se caracterizan por mostrar un pequeño cuando se sitúa el ratón sobre ellos sin pulsarlo y por ejecutar una función al pulsar sobre ellos. El fragmento de código para crear una barra de herramientas como la descrita quedaría de la forma siguiente:

```
hTBWnd= CreateToolbarEx( hWnd, WS_VISI-
      BLE | WS_CHILD | WS_BORDER |
      TBSTYLE_TOOLTIPS, IDTB_TOOL-
      BAR, NUMBUTTONS, hInst, IDTB_BIT-
      MAP, TBButton, NUMBUTTONS, 0, 0,
      16, 16, sizeof(TBBUTTON));
```

El ejemplo mostrado generará una pequeña barra de herramientas sombreada con los gráficos (iconos) indicados. Por otro lado, **TDTB. TOOLBAR** es el nombre identificativo que se ha dado a la barra. El número de elementos se define en **NUMBUTTONS** (en este caso 3). Por otra parte **IDTB.BITMAP** es la instancia de módulo con el archivo que contiene el recurso del mapa de bits (gráfico).

A los menús y sus opciones se les pueden definir como teclas de acceso rápido

TBBUTTONS contiene la dirección del *array* que contiene los textos descriptivos de cada opción de la barra de herramientas.

Por último, decir que también debe indicarse el tamaño y posición del

Listado 1. Programa completo con ejemplos de uso de menús y barras de herramientas.

```
fichero 1: <menutool.h>
#define IDTB_BITMAP     101
#define IDTB_TOOLBAR 102

#define IDM_SMALL       201
#define IDM_MEDIUM      202
#define IDM_LARGE       203

#define IDM_RED         301
#define IDM_GREEN       302
#define IDM_BLUE        303

fichero 2: <menutool.rc>
#include "windows.h"
#include "menutool.h"

IDTB_BITMAP BITMAP
"menutool.bmp"

Alteration MENU
 {
  POPUP "&Ellipse_size"
   {
    MENUITEM "&Small",
ID_SMALL
    MENUITEM "&Medium",
IDM_MEDIUM
    MENUITEM "&Large",
IDM_LARGE
   }
  POPUP "&Background_Color"
   {
    MENUITEM "&RED\tF1",
IDM_RED
    MENUITEM "&GREEN\tF2",
IDM_GREEN
    MENUITEM "&BLUE\tF3",
IDM_BLUE
   }
 }

Alteration ACCELERATORS
  {
   VK_F1, IDM_RED,      VIRTKEY
    VK_F2, IDM_GREEN,  VIRTKEY
    VK_F3, IDM_BLUE,     VIRTKEY
  }

fichero 3: <menutool.c>
#include <windows.h>
#include <commctrl.h>
#include "menutool.h"
#define NUMBUTONS 3

LRESULT CALLBACK WndProc(
HWND, UINT, WPARAM, LPARAM);
void DefToolbar( );

char szProgName[] = "ProgName";
char szApplName[] = "Alteration";
static int iSize = 30;
static int iColor;

TBBUTTON TButtons[NUMBUT-
TONS];
HWND hTBWnd;

int WINAPI WinMain(HINSTANCE
hInst, HINSTANCE hPreInst, LPSTR
lpszCmdLine, int nCmdShow)
 {

  HWND hWnd;
   HANDLE hAccel;
   MSG lpMsg;
   WNDCLASS wcApp;

 wcApp.lpszClassName  = szProgName;
  wcApp.hInstance        = hInst;
  wcApp.lpfnWndProc     = WndProc;
  wcApp.hCursor           =
LoadCursor(NULL, IDC_ARROW);
  wcApp.hIcon               = NULL;
  wcApp.lpszMenuName = szApplName;
  wcApp.hbrBackground =
GetStockObject(WHITE_BRUSH);
  wcApp.style               = CS_HRE-
DRAW | CS_VREDRAW;
  wcApp.cbClsExtra       = 0;
  wcApp.cbWndExtra     = 0;
  If(!RegisterClass (&wcApp))
   Return 0;

  hWnd = CreateWindow (szProgName,
"Menús y barras de herramientas",
WS_OVERLAPPEDWINDOW, 0, 0,
640, 480, (HWND) NULL, (HMENU)
NULL, (HANDLE) hInst, (LPSTR)
NULL);

hAccel = LoadAccelerators(hInst,
szApplName);

DefToolbar();
hTBWnd = CreateToolbarEx(hWnd,
WS_VISIBLE | WS_CHILD | WS_
BORDER | TBSTYLE_TOOLTIPS,
IDTB_TOOLBAR, NUMBUTTONS,
hInst, IDTB_BITMAP, Tbuttons, NUM-
BUTTONS, 0, 0, 16, 16,
sizeof(TBBUTTON));
ShowWindow( hWnd, nCmdShow);
UpdateWindow (hWnd);

while(GetMessage(&lpMsg, NULL, 0,
0)) {
  if(!TranslateAccelerator(hWnd, hAccel,
&lpMsg))
     {
      TranslateMessage(&lpMsg);
      DispatchMessage(&lpMsg);
     }
  }
  return(lpMsg.wParam);
}

LRESULT CALLBACK WINAPI
WndProc (HWND hWnd, UINT messg,
WPARAM wParam, LPARAM lParam)
 {
  HDC hdc;
  PAINTSTRUCT ps;
  LPTOOLTIPTEXT ToolTipStr;
  HMENU hmenu;
  static int xClientView, yClientView;
  static int iColorValue[3][3]=
{ 255, 0, 0      // Color rojo
0, 255, 0      // Color verde
0, 0 , 255 }; // Color azul

   switch (messg)
    {
     case WM_COMMAND:
     switch (LOWORD(wParam))
      {
       case IDM_SMALL:
           iSize = 20;
           break;
       case IDM_MEDIUM:
           iSize = 50;
           break;
       case IDM_LARGE:
           iSize =100;
           break;
       case IDM_RED:
       case IDM_GREEN:
       case IDM_BLUE:
            hmenu= Getmenu(hWnd);
            CheckMenuItem(hmenu,
iColor, MF_UNCHECKED);
            iColor = LOWORD(wParam);
            CheckMenuItem(hmenu,
iColor, MF_CHECKED);
            SetClassLong(hWnd,
GCL_HBRBACKGROUND,
```

Listado 1. Programa completo con ejemplos de uso de menús y barras de herramientas. (Continuación).

```
            (LONG) CreateSolidBrush
(RGB (iColorValue[iColor-DM_RED][0],
iColorValue[iColor-IDM_RED][1],
iColorValue[iColor-IDM_RED][2])));
            break;
        default:
            break;
        }
        InvalidateRect(hWnd, NULL,
TRUE);
        break;

    casse WM_SIZE:
        xClientView = LOWORD(lParam);
        yClientView = HIWORD(lParam);
        break;

    case WM_NOTIFY:
        ToolTipStr = ((LPTOOLTIPTEXT)
lParam;
        if(ToolTipStr -> hdr.code ==
TTN_NEEDTEXT)
        switch( ToolTipStr -> hdr.idFrom)
            {
            case IDM_RED:
                ToolTipStr -> lpszText =
"Rojo";
                break;
            case IDM_GREEN:
                ToolTipStr -> lpszText =
"Verde";
                break;
            case IDM_BLUE:
                ToolTipStr -> lpszText =
"Azul";
                break;
            }
        break;
    case WM_PAINT:
        hdc = BeginPaint( hWnd, &ps);

        SetMapMode(hdc,
MM_ISOTROPIC);

        SetWindowExtEx(hdc, 500, 500,
NULL);
        SetViewportExtEx(hdc,
xClientView, -yClientView, NUL);
        SetViewportorgEx(hdc,
xClientView/2, yClientView/2, NULL);

        Ellipse( hdc, -(iSize*2), -Size,
iSize*2, iSize);

        TextOut( hdc, -(iSize/2),
(iSize*2)+20, "Cambios", 11);

        ShowWindow(hTBWnd,
SW_SHOW);
        UpdateWindow(hTBWnd);

        ValidateRect(hWnd, NUL);
        EndPaint(hWnd, &ps);
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;

    default:
        return(DefWindowProc(hWnd,
messg, Wparam, lParam));
    }
    return(0);
}
void DefToolbar( )
{

    TBButtons[0].iBitmap = 0;
    TBButtons[0].idCommand=
IDM_RED;
    TBButtons[0].fstate=
 TBSTATE_ENABLED;
    TBButtons[0].fsStyle=
TBSTYLE_BUTTON;
    TBButtons[0].dwData= 0L;
    TBButtons[0].iString= 0;

    TBButtons[0].iBitmap = 1;
    TBButtons[0].idCommand=
IDM_GREEN;
    TBButtons[0].fstate=
 TBSTATE_ENABLED;
    TBButtons[0].fsStyle=
TBSTYLE_BUTTON;
    TBButtons[0].dwData= 0L;
    TBButtons[0].iString= 0;

    TBButtons[0].iBitmap = 2;
    TBButtons[0].idCommand=
IDM_BLUE;
    TBButtons[0].fstate=
TBSTATE_ENABLED;
    TBButtons[0].fsStyle=
TBSTYLE_BUTTON;
    TBButtons[0].dwData= 0L;
    TBButtons[0].iString= 0;

}
```

primer icono que aparecerá en la barra (el resto usarán el mismo tamaño).

CÓDIGO DE MANEJO DE BARRAS

Como se ha dicho, cada opción disponible en la barra de herramientas puede tener un texto descriptivo asociado que aparecerá al pasar por encima de los iconos. Estos textos descriptivos, se pueden indicar en el propio código del programa. A continuación se muestra un fragmento de código relativo a esta función:

```
if(ToolTipStr -> hdr.code== TTN_NEEDTEXT)
    switch( ToolTipStr -> hdr.idfFrom)
        {
        case IDM_RED:
            ToolTipStr -> lpszText= "Red";
            break;
        case IDM_GREEN:
            ToolTipStr -> lpszText = "Green";
            break;
...
...
...
```

Como se puede comprobar, con una simple instrucción *switch* se pueden manejar los mensajes *WM_NOTIFY* generados para cada color. En cuanto a *ToolTipStr*, decir que se trata de la cadena de la etiqueta que va a colocarse en el cuadro.

Para crear una barra de herramientas solamente tenemos que llamar a la función CreateToolbar()

¿Cómo se definen los botones de la barra? Como es lógico, cada botón de la barra de herramientas posee varios atributos y parámetro asociados que lo definen. Para definir los botones se usa la función *DefToolbar()*. Para el ejemplo dado hasta ahora, donde tenemos que definir tres botones, necesitaríamos el código siguiente: (sólo se incluye el código para dos botones, ya que es similar en los tres casos):

```
void DefToolbar( )
 {
 TBButtons[0].iBitmap = 0;
 TBButtons[0].idCommand= IDM_RED;
 TBButtons[0].fstate= TBSTATE_ENABLED;
 TBButtons[0].fsStyle= TBSTYLE_BUTTON;
 TBButtons[0].dwData= 0L;
 TBButtons[0].iString= 0;

 TBButtons[0].iBitmap = 1;
 TBButtons[0].idCommand= IDM_GREEN;
 TBButtons[0].fstate= TBSTATE_ENABLED;
 TBButtons[0].fsStyle= TBSTYLE_BUTTON;
 TBButtons[0].dwData= 0L;
 TBButtons[0].iString= 0;
 ...
 ...
}
```

De los parámetros que se definen para cada botón debemos indicar las siguientes cualidades:

- *TBButtons[0].iBitmap* se usa para indicar qué gráfico usa el botón para mostrarse.
- *TBButtons[0].idCommand* se usa para definir el color asociado al botón.
- *TBButtons[0].fsState= TBSTATE_ENABLED* hace que sea posible usar el gráfico asociado.
- *TBButtons[0].fsStyle= TBSTYLE_BUTTON* se usa para utilizar el estilo de botón típico de una barra de herramientas.
- *TBButtons[0].dwData* asocia un dato al botón.
- *TBButtons[0].iString* es un valor de cadena.

UN PROGRAMA DE EJEMPLO COMPLETO

Con todo lo explicado hasta ahora en este articulo, el lector ya está en posición de poder entender el funcionamiento del código correspondiente a un programa entero que haga uso de menús y barras de herramientas. En el listado 1 se ha incluido un ejemplo completo. Debido a que el programa usa menús, barras, gráficos y otros recursos, ha sido necesario la utilización de más de un fichero para ser definido, y por ello en el listado se incluye el contenido correspondiente a todos los ficheros que se necesitan para poder generar el ejecutable.

FUNCIONES NUEVAS QUE APARECEN

Una vez examinado a fondo el código del ejemplo, el lector habrá percibido que en el programa s aparecen unas cuantas funciones que no han sido todavía explicadas todavía. Para que el lector no tenga ningún problema se detallan a continuación:

- *GetMenu():* Esta función devuelve el *handle* asociado al menú que tenemos abierto en un momento determinado. Con este *handle* se podrán ejecutar después todas las funciones de manejo de menús, las cuales lo necesitan para operar (al igual que por ejemplo las funciones de fichero necesitan el *handle* que se obtiene al abrir un fichero del disco).
- *CheckMenuItem():* Permite poner o quitar las marcas de verificación que pueden tener algunas opciones de un menú (*MS_UNCHECKED* la elimina y *MS_CHECKED* la coloca).
- *SetClassLong():* Esta función se usa para poder cambiar el color de fondo del programa.
- *LoadAccelerators():* Permite las teclas de aceleración en los menús, las cuales deben ser cargadas previamente con esta función.
- *TranslateAccelerator():* Convierte los aceleradores de teclado en mensajes *WN_COMMAND*, de tal forma que podrán ser procesados posteriormente.
- *SetMapMode(), SetWindowExtEx(), SetViewportExtEx()* y *SetViewportOrgEx()*: Estas funciones se utilizan para poder cambiar el modo de mapeado predeterminado, para poder establecer las extensiones de la ventana y para poder reestablecer de nuevo el origen de la ventana. Todo ello en conjunto, hace que la figura que aparece en pantalla (la elipse del ejemplo), se adapte automáticamente al tamaño de la ventana en todo momento.

CONCLUSIÓN

Poder crear menús de opciones y barras de herramientas eran dos de los últimos temas más importantes que quedaban por explicar de la programación en C para Windows. Ahora que el lector ya tiene casi toda la base asimilada, damos por finalizada esta serie de artículos que espero os hayan servido de ayuda.

Herramientas para construir un cortafuegos (II)

Juan José Taboada León (taboada@uhu.es) y José Juan Mora Pérez (albeniz@uhu.es)

En el artículo anterior se introdujeron los conceptos necesarios para poder abordar las tareas de diseño de un sistema seguro utilizando Firewalls, analizando diferentes topologías. Ahora veremos algunas soluciones software para implantar cortafuegos en nuestro sistema, así como la forma más idónea de gestionar su configuración.

Con el fin de facilitar la tarea a la hora de practicar con el software requerido en este caso, nos moveremos dentro del marco del software *freeware*. De esta manera el lector podrá ensayar todas las características del software sin restricciones de ningún tipo. En este sentido hablaremos de herramientas como *TCP Wrapper*, *FWTK* de *TIS*, etc.

Antes de comenzar con las distintas herramientas disponibles para construir un cortafuegos, veremos la forma en que los sistemas Unix tratan las peticiones de conexión.

INETD

Cuando deseamos disponer de una máquina que ofrezca una serie de servicios a una red, ya sea a una intranet perteneciente a nuestra organización o a la propia Internet, dichos servicios serán ofrecidos por un servidor. Entenderemos como servidor el software encargado de tratar las peticiones de servicios y no la máquina en sí. De esta forma, a lo largo del artículo utilizaremos esta palabra para referirnos al software, mientras que para referirnos a la máquina en la que se encuentra ubicado haremos uso de la palabra *host*. En un mismo *host* podremos tener varios servidores, que pueden trabajar para el mismo servicio o para servicios distintos.

Una vez aclarado este concepto, vamos a comenzar analizando las distintas formas que tiene un servidor de ejecutarse en un *host*. En los sistemas UNIX existen dos formas distintas, en segundo plano (como demonio) o bajo demanda.

Como demonio, el servidor, normalmente y por comodidad, es lanzado a la hora de arrancar el sistema y permanece ejecutándose en segundo plano todo el tiempo que el sistema esté funcionado. En la otra forma que tiene un servidor de ejecutarse, bajo demanda, el servidor sólo se ejecuta cuando el sistema UNIX recibe una petición.

Es obvio que un servidor trabajando en segundo plano (como demonio) es mucho más rápido a la hora de atender las peticiones, ya que se encuentra ejecutándose cuando recibe éstas. En cambio trabajando bajo demanda, existe un elemento intermedio que será el encargado de ejecutar el servidor cada vez que se reciba una petición de servicio.

Aunque la velocidad de respuesta del servidor ante una petición pueda suponer una razón lo suficientemente

importante para que el servidor esté ejecutándose en segundo plano todo el tiempo, sobre todo en aquellos *host* dedicados a dar un solo servicio, presenta una desventaja con respecto a la seguridad de nuestro sistema.

En un mismo host podremos tener varios servidores, que pueden ser para el mismo servicio o para servicios distintos

Hemos citado la existencia de un elemento que actúa entre la recepción de solicitudes de servicio y el servidor, dicho elemento pertenece al sistema UNIX y consiste en el demonio **inetd.** Este demonio puede ser configurado para aumentar la seguridad de nuestro sistema, aunque su propósito original no fue el de aumentar la seguridad de los sistemas UNIX, sino más bien el rendimiento. Estos lo consigue evitando que en un *host* estén ejecutándose una serie de servidores, los cuales no tienen un uso intensivo, tales como las conexiones *TELNET*, *FTP*, *FINGER*, etc.

Todos los servidores de servicios disponibles en un sistema UNIX, tales como *TELNET*, *FTP*, *FINGER*, *HTTP*, etc., tienen asociado un puerto a través del cual se realiza la comunicación con el exterior. Para definir en qué puerto va a trabajar un servicio determinado debemos especificarlo en el fichero **/etc/services,** tal y como aparece en la Tabla 1.

En este fichero reflejado en la Tabla 1 se define qué puerto va a utilizar cada servicio, de tal forma que podemos afirmar que su función consiste en determinar los puertos que están abiertos y los que se encuentran cerrados, así como para asignarlos un nombre. De esta manera, cuando **inetd** recibe una petición de conexión a un servidor determina el puerto por el que se efectúa la petición. A continuación busca en el fichero **/etc/services** a qué servicio pertenece este puerto y una vez que obtiene su nombre, examina dentro del fichero **/etc/inetd.conf** la línea que define la forma en la que se va a tratar la petición.

Tabla 1. Fichero /etc/services.

```
#
# from: @(#)services  5.8 (Berkeley) 5/9/91
# $Id: services,v 1.9 1993/11/08 19:49:15 cgd Exp $
#
discard       9/tcp     sink null
discard       9/udp     sink null
systat        11/tcp    users
daytime       13/tcp
daytime       13/udp
netstat       15/tcp
qotd          17/tcp    quote
ftp           21/tcp
telnet        23/tcp
smtp          25/tcp    mail
time          37/tcp    timserver
time          37/udp    timserver
finger        79/tcp
www           80/tcp    http     # WorldWideWeb HTTP
www           80/udp             # HyperText Transfer Protocol
link          87/tcp    ttylink
sftp          115/tcp
netbios-ns    137/tcp                  # NETBIOS Name Service
netbios-ns    137/udp
netbios-dgm   138/tcp                  # NETBIOS Datagram Service
netbios-dgm   138/udp
netbios-ssn   139/tcp                  # NETBIOS session service
netbios-ssn   139/udp
```

El fichero **/etc/inetd.conf**, de la Tabla 2, es el fichero de configuración del demonio **inetd**. Está constituido por una serie de filas, cada una de las cuales define cual será el comportamiento de **inetd** ante una determinada solicitud de servicio.

Vamos a ver las distintas partes que componen cada una de las líneas de este fichero, tomando como ejemplo la línea que define el comportamiento de **inetd** ante una petición para una conexión *TELNET* :

```
telnet stream tcp nowait root
      /usr/local/bin/in.telnetd in.telnetd
```

Con el primer campo, **telnet**, existe cierta confusión, ya que en un principio al ver los ficheros del sistema podemos caer en el error de considerar que este campo sirve para identificar el servicio que se desea configurar. Nada más lejos de la realidad, puesto que hemos visto que en el fichero **/etc/services** están definidos todos los puertos que nuestro sistema va a utilizar y les hemos dado un nombre, y es este nombre el que utilizamos como primer campo en la línea de arriba, es decir en el fichero **/etc/services** debe existir una línea como la siguiente:

```
telnet 23/tcp
```

Por lo tanto en las dos líneas anteriores podríamos cambiar **telnet**

Tabla 2. Fichero /etc/inetd.conf.

```
#
#ident "@(#)inetd.conf    1.22    95/07/14 SMI"        /* SVr4.0 1.5 */
#
# Configuration file for inetd(1M).  See inetd.conf(4).
#
# To re-configure the running inetd process, edit this file, then
# send the inetd process a SIGHUP.
#
# Syntax for socket-based Internet services:
# <service_name> <socket_type> <proto> <flags> <user> <server_pathname> #<args>
#
# Syntax for TLI-based Internet services:
#
#  <service_name> tli <proto> <flags> <user> <server_pathname> <args>
#

    ..................

# Ftp and telnet are standard Internet services.
#
ftp     stream tcp    nowait root    /usr/sbin/in.ftpd         in.ftpd
telnet stream tcp nowait root /usr/sbin/in.telnetd  in.telnetd

# Shell, login, exec, comsat and talk are BSD protocols.
#

shell  stream      tcp     nowait root    /usr/sbin/in.rshd        in.rshd
login  stream      tcp     nowait root    /usr/sbin/in.rlogind     in.rlogind
talk    dgram      udp     wait     root    /usr/sbin/in.talkd       in.talkd
#
# Must run as root (to read /etc/shadow); "-n" turns off logging in
# utmp/wtmp.
#

uucp  stream       tcp     nowait root    /usr/sbin/in.uucpd       in.uucpd

#
# Tftp service is provided primarily for booting.  Most sites run this
# only on machines acting as "boot servers."
#
#tftp   dgram       udp     wait     root    /usr/sbin/in.tftpd       in.tftpd -s /tftpboot
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
#
#finger stream tcp nowait nobody /usr/sbin/in.fingerd      in.fingerd
```

por cualquier otra palabra y nuestro sistema podría recibir peticiones *TELNET* normalmente.

Puede parecer que lo anterior es algo bastante obvio y que no necesita de aclaración, pero el problema viene cuando administradores noveles, se encuentran con la necesidad de implantar varios servidores para el mismo servicio en un mismo *host*, cada uno en un puerto distinto.

El segundo campo, es el tipo de *socket* que se utiliza para el servicio, ya sea de flujo o de datagramas.

El tercer campo define el tipo de protocolo que utilizará el servidor, en nuestro ejemplo se utilizará **tcp.**

El campo *nowait*, indica al demonio **inetd**, que no debe esperar a que termine una conexión para atender otras.

El quinto campo, *root*, define los permisos con los que se ejecutará el proceso que atenderá la solicitud. Por último, especificamos cual va a ser el software encargado de tratar la solicitud y los parámetros que necesita.

Por lo tanto, con lo visto hasta ahora, podemos describir qué ocurre en nuestro sistema UNIX cuando recibe una petición de conexión a un servicio determinado:

- **Inetd** despierta, al detectar una petición de conexión por alguno de los puertos descritos en el fichero **/etc/services**.
- **Inetd** busca en el fichero **/etc/services** el nombre que tiene asociado el puerto.
- Con el nombre del puerto, **inetd** busca en el fichero **/etc/inetd.conf**, la línea para tratar las peticiones que lleguen por dicho puerto.
- Lanza un proceso, el cual tratará la petición de conexión, ejecutando el quinto campo de la línea de **/etc/inetd.conf**, según hayamos especificado en los otros campos, *stream, tcp, nowait, root.*

Ya hemos visto la independencia entre servicios y puertos, aunque de forma estándar suele asignarse un determinado servicio a un determinado puerto como podemos ver en la Tabla 3, ya que la mayoría de aplicaciones utilizarán estos puertos para comunicarse con un cliente.

Nosotros como administradores podemos decidir cambiar el puerto que va a utilizar un servidor, pero si hacemos esto tenemos que asegurarnos siempre que el cliente dirige su solicitud al nuevo puerto, ya que se nos podría presentar una situación como la siguiente: supongamos que en nuestro sistema UNIX se encuentra ejecutándose un servidor *HTTP*. El demonio de este servidor es **httpd** y en la configuración hemos definido que el puerto que va a utilizar será **8080**, y es por este puerto por el cual **httpd** se comunicará con sus clientes.

Inetd puede configurarse para aumentar la seguridad de nuestro sistema

Si un usuario intenta acceder a nuestra página web, su solicitud fallará, ya que por defecto los navegadores mandan sus solicitudes hacia el puerto **80.** En nuestro caso **httpd** estará escuchando por el puerto **8080**, así que únicamente los usuarios que conozcan el número del puerto por el que está escuchando nuestro servidor **httpd** podrán acceder a sus páginas web. Para indicarle al navegador que debe dirigir sus peticiones a un puerto distinto al 80, tendremos que utilizar la siguiente nomenclatura:

```
http://www.mipagina.com: 8080
```

Para trabajar con un cortafuegos es condición necesaria conocer la forma en que trabaja el demonio **inetd**, ya que será éste el que tendremos que reconfigurar, cambiando puertos, eliminado algunos servicios como por ejemplo el problemático **finger** o decidiendo cuales serán las acciones que realizará ante la solicitud de un determinado servicio.

Por lo tanto si queremos que nuestro *host* sea un lugar seguro estaremos obligados no solo a trabajar con software para construir cortafuegos, como veremos más adelante, sino también a conocer a fondo como trabajan los sistemas UNIX y sobre todo los demonios como **inetd** y **syslogd**. En este artículo hemos analizado **inetd** y por razones de espacio, el análisis de **syslogd** lo aplazaremos para el siguiente, aunque adelantaremos que se trata de un demonio encargado de registrar todo lo que ocurre en nuestro sistema. Más de un *hacker* despistado ha sido descubierto gracias a este demonio.

Tabla 3. Puertos.

Puerto	Servicio	Descripción
13	fecha	Fecha y hora de la máquina
15	netstat	Información sobre la red
21	ftp	Transferencia de ficheros
23	telnet	Conexión remota
25	smpt	Correo
53	domain	Nombre de la máquina
79	finger	Información sobre usuarios
80	http	Servidor Web
110	pop	Correo
513	rlogin	Conexión remota
520	route	Información de rutado

Cada vez que realicemos alguna modificación en el fichero de configuración **/etc/inetd.conf**, los cambios no afectarán al sistema hasta que vuelva a inicializarse el demonio o hasta que mandemos una señal *HUP* para que se reinicie **inetd:**

```
kill -HUP pid_inetd
```

TCP WRAPPERS

Esta herramienta no la podemos considerar como una utilidad para confeccionar un cortafuegos, pero para aquellos usuarios noveles en el mundo de la seguridad, es una buena idea comenzar con ella debido a su gran sencillez conceptual y a la facilidad de configuración e instalación que supone en nuestro sistema.

Ya conocemos como trabaja **inetd**, recordemos que una de las ventajas que poseía era que permitía controlar la forma en la que nuestro sistema aceptaba las peticiones de servicios. Pues bien *TcpWrapper* aprovecha este esquema para realizar una filtración de las solicitudes de servicios que reciba nuestro sistema. Por lo tanto, *TcpWrapper* trabaja entre **inetd** y el programa encargado de tratar la conexión. Así una vez que se haya instalado y configurado esta herramienta, cada vez que nuestro sistema reciba una petición de conexión a alguno de los servicios de los que disponga, **inetd** no ejecutará el programa encargado de tratar la conexión sino que ejecutará **tcpd** (es el software de TcpWrapper), y como parámetro se utilizara el nombre del programa encargado de tratar la conexión.

Todos los servidores tienen asociado un puerto

Podemos comprobar que el esquema es bastante sencillo, por un lado tenemos el demonio **inetd** que recibe las peticiones de servicio, seguidamente se ejecuta **tcpd**, para que se realice una filtración y por último se ejecuta el programa o la acción que corresponda para tratar la petición de servicio.

Tabla 4. Reglas para TcpWrapper.

```
#
# Fichero : /etc/hosts.allow
#
telnetd : host1.com, host3.com
ftpd    : host3.com

#
# Fichero : /etc/hosts.deny
#
ftpd    : host2.com, 111.112
httpd   : host3.com
ALL     : 112.222
```

Si nuestro sistema es Linux, es probable que tengamos instalado TcpWrapper por defecto. Es posible efectuar esta comprobación editando el fichero **/etc/inetd.conf** y buscando alguna línea como la siguiente:

```
finger stream tcp nowait nobody /usr/etc/tcpd
    in.fingerd
```

Esta línea indica que el servicio *finger* está siendo filtrado con *TcpWrapper*. Tanto si lo tenemos instalado, en cuyo caso quizás deseemos configurar otros servicios para que también sean filtrados, como si vamos a instalarlo en nuestro sistema resulta importante conocer la forma en la que trabaja TcpWrapper:

- El sistema recibe una petición de conexión para alguno de los servicios disponibles en nuestro sistema, por ejemplo una petición WWW.
- **Inetd** busca en su fichero de configuración y ejecuta la línea dedicada a dicho servicio.
- **Inetd** ejecuta **tcpd** y como parámetro (**tcpd** únicamente puede recibir un parámetro) se le pasa el nombre del programa encargado de tratar la conexión.
- **Tcpd** busca en sus ficheros de configuración, **/etc/hosts.allow** y **/etc/hosts.deny**, la aparición de alguna regla que sea aplicada al servicio en cuestión.
- Según los criterios que hayamos reflejado en las reglas, una conexión se aceptará o se rechazará. En el primer caso **tcpd** ejecutará el programa que se encargue de tratar la conexión.

Como hemos mencionado anteriormente, la configuración de *TcpWrapper*, se basa en dos archivos:

- **-/etc/hosts.allow**, en este fichero aparecerán la lista de direcciones IP que tienen permitido el acceso a nuestra máquina.
- **-/etc/hosts.deny**, al contrario que el anterior, este fichero contiene la lista de direcciones IP que no tienen permitido el acceso a nuestra máquina.

Ambos están constituidos por reglas. Si un servicio, aparece en una regla en el fichero **/etc/hosts.allow** y el mismo servicio, aparece en otra regla en el fichero **/etc/hosts.deny**, la regla del este último fichero, prevalecerá sobre las del primero. Las reglas presentan el siguiente formato:

```
Servicio1, Servicio2, ... : HOST1, HOST2, IP3,
    IP4, ....

Servicio1, Servicio2, ...,
```

Consiste en una lista de servicios a los cuales se les va a aplicar esta regla, y que son identificados mediante los nombres utilizados en el fichero **/etc/services**. A continuación, separada por dos puntos, se da una lista de direcciones, las cuales pueden ser nombres de máquinas, IP o dominios.

Podemos utilizar la palabra *ALL*, tanto para definir servicios como direcciones. Si una regla contiene esta palabra en la lista de servicios, **tcpd** interpreta que debe aplicar esta regla a todos los servicios disponibles en la máquina. Si la palabra *ALL* se encuentra en la lista de direcciones, se interpretará que esta regla es aplicada a todas las direcciones.

Vamos a plantear un ejemplo de posibles criterios que *TcpWrapper* debería aplicar a las conexiones:

- **host1.com** tiene acceso únicamente si realiza conexiones *TELNET*.
- **host2.com** no tiene acceso utilizando *FTP*.
- **host3.com** tiene acceso a *FTP* y *TELNET*, pero no tiene acceso a *HTTP*.
- Todas las máquina del dominio *111.112.*.** tienen el acceso denegado si utilizan *FTP*.
- Todas las máquina del dominio *112.222.*.** tienen acceso denegado a todos los servicios de los que disponga nuestra máquina.

CONCLUSIONES

Después de haber introducido los conceptos sobre Firewalls en el artículo anterior, en éste se ha descrito la forma de trabajar de **inetd**, con el objetivo de conocer cómo se tratan las peticiones de conexión en nuestro sistema. Este conocimiento permitirá diseñar un perfil de seguridad lo suficientemente restrictivo. A continuación se han ofrecido algunas nociones para iniciar la instalación y configuración de nuestro primer cortafuegos. En la próxima entrega trataremos con profundidad una herramienta más completa, que además nos permitirá configurar el servidor como un servidor Proxy.

Reconocimiento de voz (IV)

Constantino Sánchez Ballesteros (constantino@redestb.es)

Este mes veremos la programación de una parte de la API Speech mediante C++. En concreto, aprenderemos a activar el programa encargado de ajustar correctamente el micrófono. Además se realizará una aplicación de dictado y texto hablado mediante Visual Basic.

Como ya comenté en un artículo anterior de la serie, la programación de la API Speech en el entorno C/C++ es un poco más complicada que la de Visual Basic pero a su vez más potente. Como novedad, los dos programas que vamos a ver este mes utilizan la última versión de la API (la 4.0). Para poder instalar correctamente las librerías de Speech en el entorno de Visual C++ 5.0 es necesario efectuar un par de modificaciones en la estructura de directorios de búsqueda del compilador.

- Debemos ir al menú Herramientas/Opciones del compilador y seleccionar la pestaña Directorios.

- Una vez hecho esto, introduciremos los directorios que contienen las librerías y los headers de la API Speech.

El asistente del micrófono es un programa que se encarga de tomar los registros de voz que provienen del micrófono para ajustar su sensibilidad correctamente. Puesto que la *API Speech* está completamente en inglés, este asistente se nos mostrará en ese idioma. Éste ya está programado por M-crosoft, por lo tanto, para utilizarlo sólo debemos llamar a unas funciones en nuestro programa.

Activando el Wizard (asistente) del micrófono, ajustaremos este dispositivo para afinar el reconocimiento de las palabras

El proyecto **MICWIZ** que vamos a ver y que incluye el CD-ROM de la revista, está realizado en Visual C++ 5.0. Existe un archivo llamado **MicWiz.mak** que se ha creado para poder utilizar el programa en otros compiladores compatibles C++. El proyecto consta de un archivo principal llamado **MicWiz.cpp** que será comentado a continuación.

En primer lugar se deben incluir los *headers* necesarios para utilizar funciones estándar del entorno Windows y la *API Speech*:

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <mmsystem.h>
#include <initguid.h>
#include <objbase.h>
#include <objerror.h>
#include <ole2ver.h>

#include <speech.h>
```

Creamos la función principal del programa (*WinMain*) para inicializarlo, dejando constancia de que en el proyecto no se utilizan las **MFC** (*Microsoft Fundation Classes*):

```
int PASCAL WinMain(HINSTANCE hInstance,
      HINSTANCE hPrevInstance,
            LPSTR lpszCmdLine, int nCmdShow)
{
  BOOL fAsk = TRUE;

  if (!stricmp(lpszCmdLine, "/n"))
    fAsk = FALSE;
```

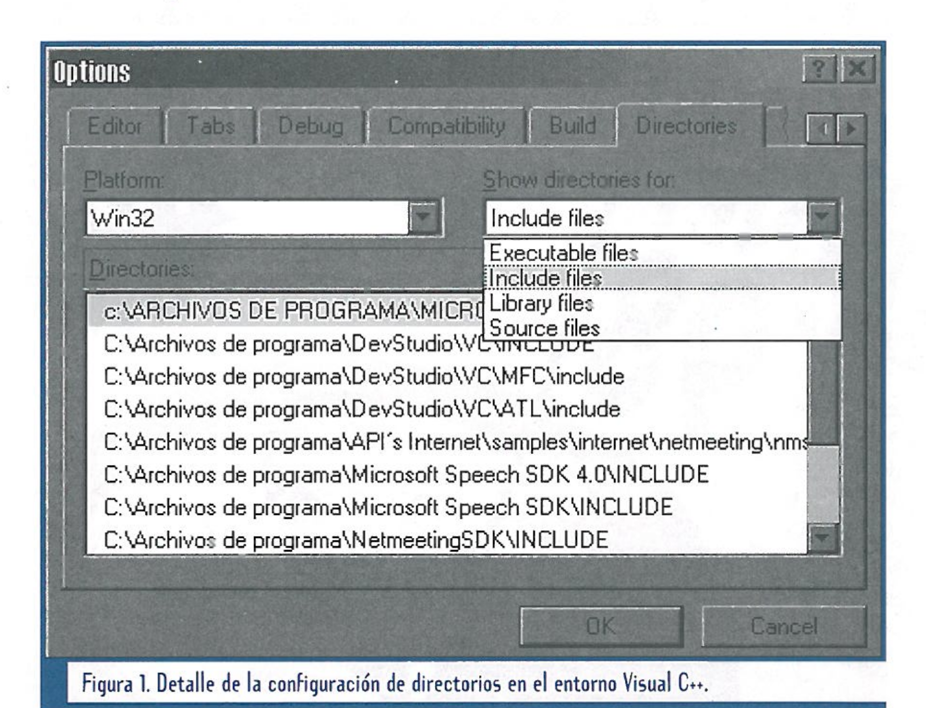


Figura 1. Detalle de la configuración de directorios en el entorno Visual C++.

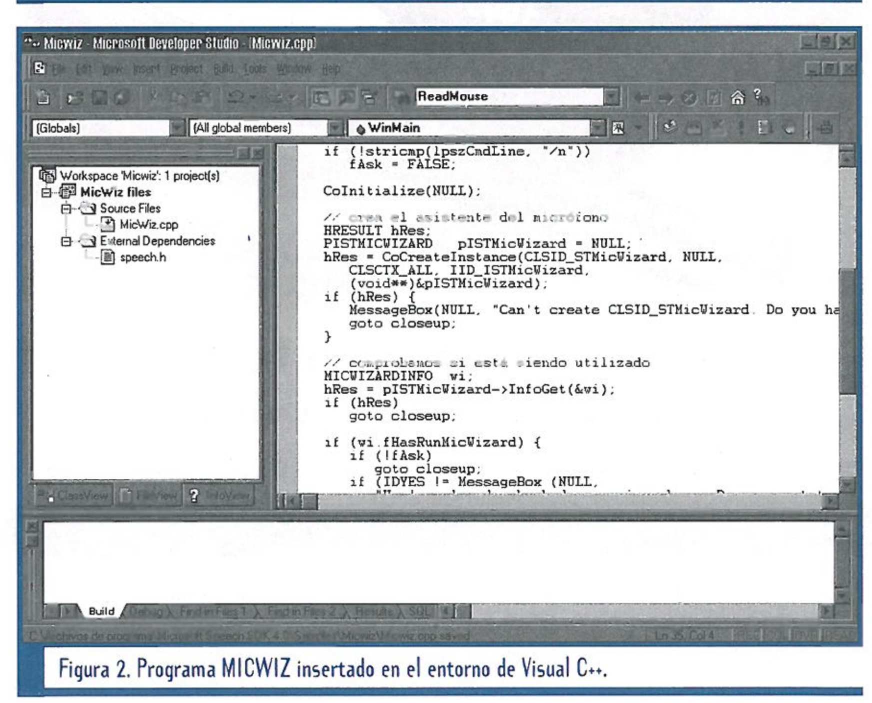


Figura 2. Programa MICWIZ insertado en el entorno de Visual C++.

Seguidamente utilizaremos la función *CoInitialize* para inicializar las librerías *OLE*:

```
CoInitialize(NULL);
```

El siguiente paso que tenemos que dar consiste en crear el asistente para el micrófono. Para ello, es necesario crear una variable de tipo *PISTMICWIZARD*. Mediante la función *CoCreateInstance* efectuamos la creación del *Wizard*:

```
HRESULT hRes;
PISTMICWIZARD  pISTMicWizard = NULL;
hRes = CoCreateInstance(CLSID_STMicWizard, NULL,
   CLSCTX_ALL, IID_ISTMicWizard,
   (void**)&pISTMicWizard);
```

En el caso de que esté instalada una versión inferior a la 4.0 se enviará un mensaje de error al usuario:

```
if (hRes) {
   MessageBox(NULL, "No se puede crear el
      asistente. Tienes instalada la versión
      3.0?", NULL, MB_OK);
   goto closeup;
}
```

El siguiente paso consiste en comprobar si el asistente está siendo utilizado actualmente o ya se abrió anteriormente para afinar la percepción por el micrófono. Para tomar información del asistente es necesario crear una variable de tipo *MICWIZARDINFO*. En ella se albergará toda la información concerniente al asistente, siendo la estructura de este tipo de variables la que se muestra en la Tabla 1.

Mediante el método *InfoGet* obtenemos en la variable *wi* de tipo *MICWIZARINFO* los datos requeridos:

```
MICWIZARDINFO  wi;
hRes = pISTMicWizard->InfoGet(&wi);
if (hRes)
   goto closeup;
```

En el caso de que se haya ejecutado anteriormente el asistente, preguntaremos al usuario si quiere volver a inicializarlo:

```
if (wi.fHasRunMicWizard)
{
   if (!fAsk)
      goto closeup;

   if (IDYES != MessageBox (NULL,
      "Ya has utilizado el asistente. Quieres volver a hacerlo?",
      "Microphone Setup Wizard",
      MB_YESNO))
      goto closeup;
}
```

El método encargado de ejecutar el asistente es *Wizard*. En la Tabla 2 se muestra el detalle de miembros del método.

```
pISTMicWizard->Wizard(NULL,
STMWU_DICTATION, WAVE_MAPPER,
16000, STMWU_CNC);
```

La etiqueta **closeup** es una bifurcación del programa para finalizar el asistente por algún motivo.

```
closeup:
  if (pISTMicWizard)
    pISTMicWizard->Release();

  CoUninitialize ();

  return 0;
}
```

PROYECTO DICTAPAD

Tal y como se reseñó en la cabecera del artículo, vamos a realizar una aplicación de dictado mediante Visual Basic. La funcionalidad de este programa nos permite efectuar dictado y que el programa lea las palabras reconocidas mediante el *engine* de texto hablado. Será necesario incluir los componentes *ActiveX* de dictado y Texto hablado para que se pueda efectuar la compilación del programa. También es requisito indispensable tener instalada la *API Speech 4.0* en nuestro ordenador.

El proyecto consta de un módulo y un formulario:

Módulo **DICTAPAD.BAS**: Define una serie de constantes para la correcta representación del formulario en pantalla.

```
Public Const SWP_NOMOVE = 2
Public Const SWP_NOSIZE = 1
Public Const FLAGS = SWP_NOMOVE Or
        SWP_NOSIZE
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Declare Function SetWindowPos Lib "user32"
    (ByVal hwnd As Long, ByVal hWndInser-
    tAfter As Long, ByVal x As Long, ByVal y
    As Long, ByVal cx As Long, ByVal cy As
    Long, ByVal wFlags As Long) As Long

Type POINTAPI
    x As Long
    y As Long
End Type

Declare Function GetCaretPos Lib "user32"
            (lpPoint As POINTAPI) As Long
```

Formulario **DICTAPAD.FRM**: Contiene un menú de opciones con diversos componentes *ActiveX* para conseguir la correcta funcionalidad del programa.

Tabla 1. Estructura de MICWIZARDINFO.

```
typedef struct
{
   BOOL          fHasRunMicWizard;
   DWORD         dwWaveInDevice;
   DWORD         dwMicrophoneType;
   WCHAR         szMicString[64];

} MICWIZARDINFO, *PMICWIZARDINFO;
```

Parámetros	Descripción
fHasRunMicWizard	TRUE si el usuario actual ya ha utilizado antes el asistente. FALSE en caso contrario.
dwWaveInDevice	Número del dispositivo de entrada que se eligió para obtener datos de sonido.
dwMicrophoneType	Tipo de micrófono.
szMicString	String que especifica el nombre del micrófono.

El parámetro dwMicrophoneType puede devolver uno de los siguientes valores:

ValorValor	dedDescripción
STMWI_UNKNOWN	Desconocido
STMWI_CLOSETALK	Micrófono de cabeza
STMWI_EARPIECE	Micrófono con un elemento al lado de la oreja.
STMWI_HANDSET	Telefonillo.
STMWI_CLIPON	Micrófono de solapa.
STMWI_DESKTOP	Micrófono de sobremesa.
STMWI_HANDHELD	Micrófono de mano (como el de los músicos).
STMWI_TOPMONITOR	Micrófono instalado encima del monitor.
STMWI_INMONITOR	Micrófono instalado en la carcasa del monitor.
STMWI_KEYBOARD	Micrófono instalado en el teclado.
STMWI_REMOTE	Micrófono por control remoto.

Definimos dos variables para representar el fichero con el que vamos a trabajar y el comienzo de lectura de una palabra:

```
Dim gThisFile As String
Dim gReadStart As Integer
```

Cuando se para de recibir audio, establecemos los botones del formulario representativos como desactivados:

```
Private Sub DirectSS1_AudioStop(ByVal hi As
        Long,
         ByVal lo As Long)
    Toolbar1.Buttons(15).Value = tbrUnpressed
    Toolbar1.Buttons(15).Image = 14
End Sub
```

Tabla 2. Miembros que componen el método Wizard.

```
Wizard
  HRESULT  Wizard (HWND hWndParent,
  DWORD dwUse = STMWU_CNC,
  DWORD dwWaveInDevice = WAVE_MAPPER,
  DWORD dwSamplesPerSec = 16000,
  DWORD dwFlags = 0)
```

La función devuelve después de que finaliza el asistente NOERROR si se concluyó bien o uno de estos valores de error:

E_INVALIDARG
E_FAIL – el usuario no tiene tarjeta de sonido que pueda grabar audio.

Parámetros	Descripción
HwndParent	Crea el asistente en la ventana indicada.
dwUse	Usa el asistente de diferentes modos: STMWU_CNC – asistente para Command and Control. Será menos estricto sobre la calidad del audio grabado. STMWU_DICTATION – asistente para dictado siendo estricto en la calidad del sonido. STMWU_NOAUTOGAIN – no utiliza autoganancia en sonido, aunque no lo desactiva. STMWU_LOWERGAIN – activa poca ganancia en el volumen del sonido.
DwWaveInDevice	Se utiliza para grabación y ajuste. Si se utiliza el valor WAVE_MAPPER y existe más de un dispositivo de sonido, el asistente preguntará al usuario qué dispositivo utilizar. La elección del usuario puede ser obtenida mediante el método InfoGet.
DwSamplesPerSec	Ratio de sampleo al que escuchará el asistente. Normalmente se establece el valor 16000.
DwFlags	Uno o más de los siguientes valores conectados por "or". STMWF_CANSKIP – si el asistente ya ha sido ejecutado por el usuario, se retornará un mensaje afirmativo. Una aplicación puede ejecutar ISTMic Wizard->Wizard() cada vez que comience, utilizando STMWF_CANSKIP, y el usuario tendrá garantizada la ejecución del asistente.

El siguiente procedimiento se encarga de seleccionra una palabra determinada para su posible modificación por parte del usuario. Para ello, será necesario obtener su longitud delimitada por un espacio " ".

```
Private Sub DirectSS1_WordPosition(ByVal hi
        As Long,
        ByVal low As Long, ByVal byteoffset As
        Long)
Dim firstspace As Integer
```

Los *strings* de Visual Basic son de tipo *Unicode*, por lo que debemos dividir el desplazamiento del *byte* por 2 para obtener el carácter. Todos los caracteres recibidos a través del habla por el micrófono son albergados en una caja de tipo *RichText* (texto enriquecido). Cuando se encuentre un espacio, terminará la longitud de la palabra:

```
RichTextBox1.selstart = byteoffset / 2 + gRe-
        adStart
firstspace = FindTextBreak(1,
        RichTextBox1.selstart, " " + vbNewLine
        + vbLf + vbCr + vbCrLf)
RichTextBox1.SelLength = firstspace - Rich-
        TextBox1.selstart

On Error GoTo done
```

Una vez obtenida la longitud de la palabra, podemos seleccionarla. Para ello, es necesario utilizar los métodos *Lock* y *Unlock*. El primero protege el texto para que nada, excepto la aplicación, pueda modificarlo. Ninguna otra función de texto funcionará hasta que se invoque al método *UnLock*.

Vdict1 es el objeto *ActiveX* que representa al módulo de dictado:

```
Vdict1.Lock
Vdict1.TextSelSet RichTextBox1.selstart, Rich-
        TextBox1.SelLength
Vdict1.Unlock
done:
End Sub
```

Inicializamos unos valores al comienzo de la carga del formulario mediante el siguiente procedimiento:

```
Public Sub Form_Load()

Dim command As String
Dim Description As String
Dim category As String
Dim action As String
Dim FLAGS As Long

On Error GoTo ErrorMessage
```

Si no se ha inicializado el *engine* de dictado, lo hacemos asignando el valor **1** al método *Initialized*:

```
If (Vdict1.Initialized = 0) Then
  Vdict1.Initialized = 1
  Vdict1.Mode = 32
End If
```

Mediante *Listen* activamos o desactivamos la sesión de dictado. Si se asigna un valor **0** la sesión no está acti-

vada; si asigna el valor **1** se establecerá la nueva sesión de dictado:

```
listen (0)
```

Inicializamos varias variables para definir el aspecto de la barra de herramientas, borrado del texto que haya en la caja *Richtext*, filtrado de archivos para su posterior selección...

```
gThisFile = ""
ResetText
Alwaysselect.Checked = True
Toolbar1.ButtonHeight = ImageList1.Image-
      Height
Toolbar1.ButtonWidth = ImageList1.Image-
      Height
Toolbar1.Height = Toolbar1.ButtonHeight
RichTextBox1.Top = Toolbar1.Top + Tool-
      bar1.Height

CommonDialog1.Filter = "Dictation
      (*.msd)|*.msd|Rich Text
      (*.rtf)|*.rtf|Text (*.txt)|*.txt"
GoTo NoError

ErrorMessage:
MsgBox "No se inicializó el engine de dictado.
      Asegúrese de que tiene uno instalado en
      el ordenador"
End

NoError:

End Sub
```

Mediante la opción de menú *Addword* agregaremos las nuevas palabras que necesitemos al diccionario del *engine*. Para ello es necesario llamar al método *LexiconDlg* del objeto **Vdict1:**

```
Private Sub Addword_Click()
On Error GoTo addend
Vdict1.LexiconDlg Form5.hwnd, "Add Word"
addend:
End Sub
```

La opción de menú **alwaysontop** es la encargada de que el usuario pueda ver la ventana del proyecto constantemente aunque en algún momento determinado pierda el *focus* a favor de otra aplicación:

```
Private Sub alwaysontop_Click()
alwaysontop.Checked = Not
      (alwaysontop.Checked)
If (alwaysontop.Checked) Then
   FloatPad (True)
Else
   FloatPad (False)
```

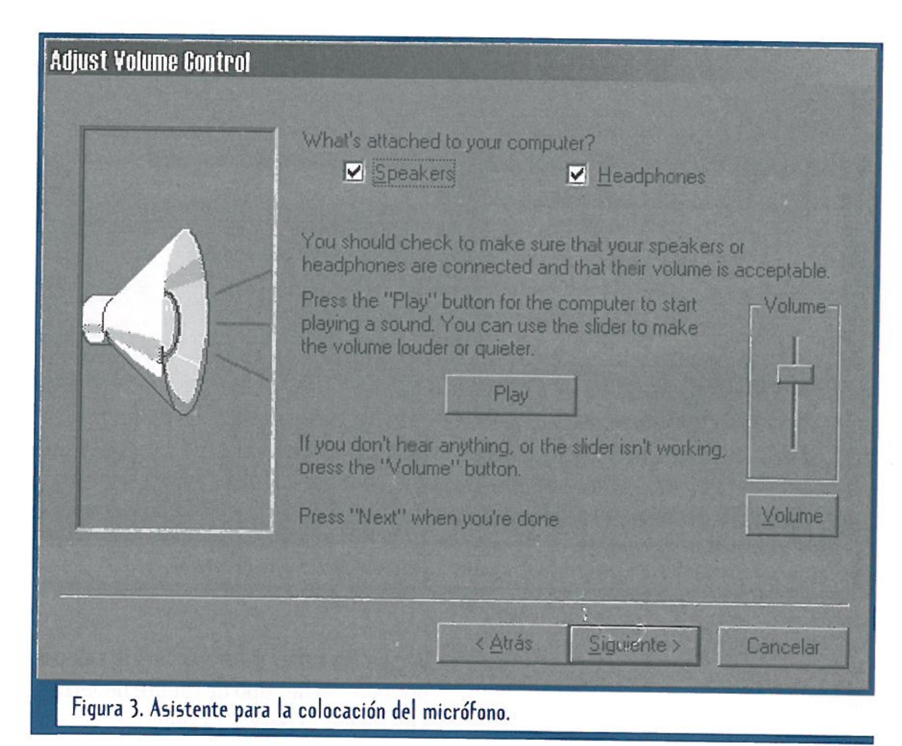


Figura 3. Asistente para la colocación del micrófono.

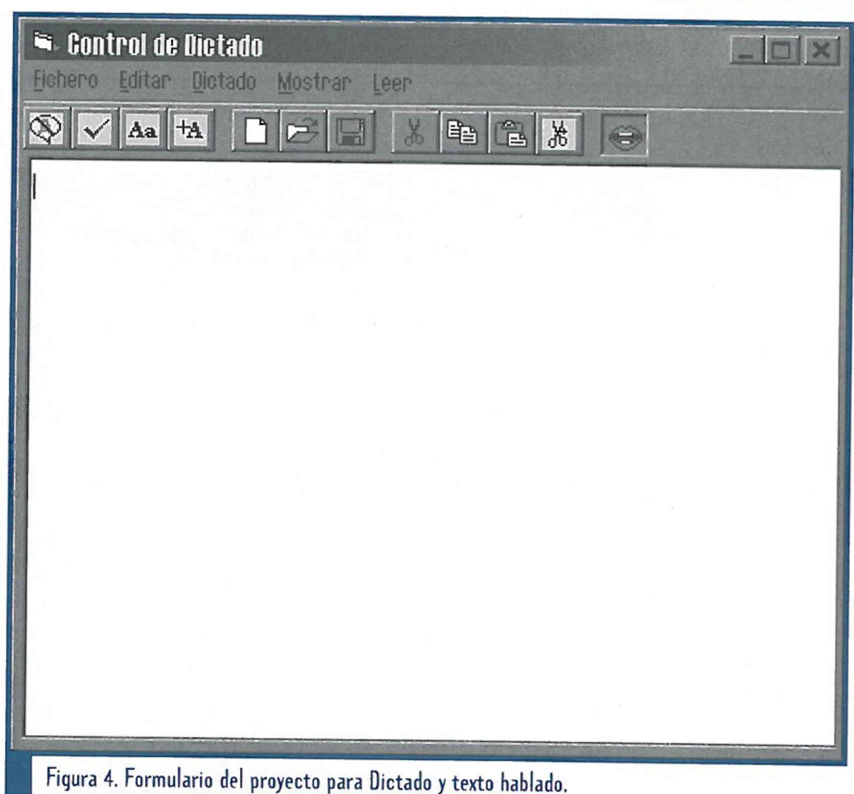


Figura 4. Formulario del proyecto para Dictado y texto hablado.

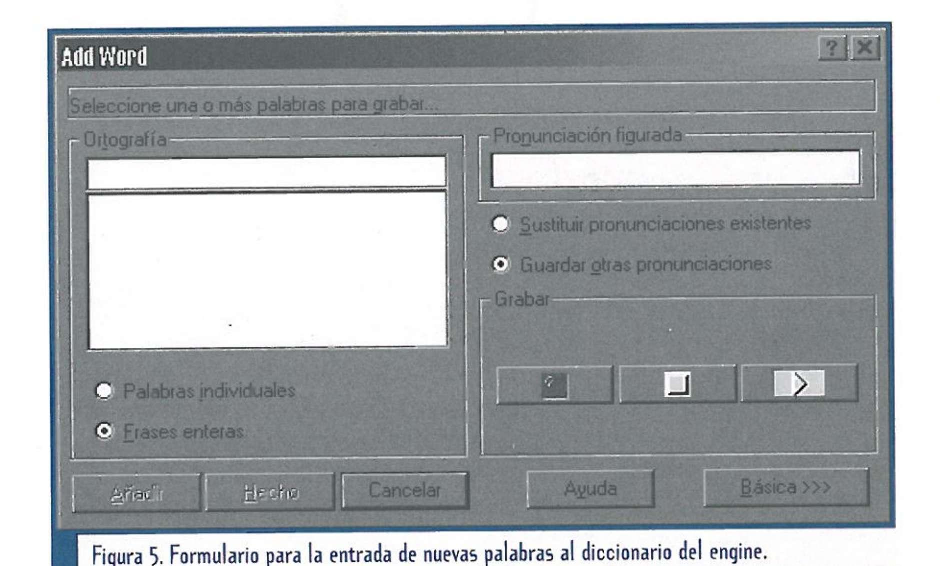


Figura 5. Formulario para la entrada de nuevas palabras al diccionario del engine.

```
End If
End Sub
```

El siguiente procedimiento se encarga de borrar todo el contenido de la caja de texto y establece los valores por defecto del tipo de fuente de letra y su tamaño (*Times* a 14 puntos):

```
Private Sub ResetText()
RichTextBox1.Font.Name = "Times New
    Roman"
RichTextBox1.Font.Size = 14
SelectAll_Click
Delete_Click

RichTextBox1.Font.Name = "Times New
    Roman"
RichTextBox1.Font.Size = 14
End Sub
```

Para poder seleccionar palabras con un único clic de ratón, se debe hacer referencia al procedimiento:

```
Private Sub Alwaysselect_Click()
Alwaysselect.Checked = Not
    (Alwaysselect.Checked)
End Sub
```

Debido al espacio, en la siguiente entrega acabaremos con este proyecto y comenzaremos otro que utilice nuevas características de la *API Speech*.

Introducción a los JavaBeans

Javier Sanz Alamillo (jsanz@a01-unix.uc3m.es)

Si has usado Visual Basic, Delphi o cualquier otro entorno visual de programación, entonces ya estás familiarizado con la noción de Bean. La idea es la misma, aunque el lenguaje de programación utilizado sea diferente. Un JavaBean es un componente software escrito en Java que puede ser utilizado en entornos visuales.

La aparición de entornos de tipo RAD (*Rapid Application Development: Desarrollo Rápido de Aplicaciones*) como Microsoft Visual Basic o Borland C++ ha hecho que los programadores se familiaricen con el concepto de componente software.

LOS COMPONENTES SOFTWARE

Los componentes software pueden ser un grupo de clases o un conjunto de funciones y estructuras de datos escritas en un determinado lenguaje de programación que ofrecen una determinada funcionalidad al programa que los utiliza. Se caracteriza por ocultar su implementación, de acuerdo a unos interfaces, y por encapsular los datos, como las clases de cualquier lenguaje de programación orientado a objetos. Estos componentes son diseñados para poder ser reutilizados sin modificación alguna e incluso para intercambiarse.

Un JavaBean es un componente software escrito en Java

Ahora bien, si utilizamos el lenguaje Java, se plantea la siguiente pregunta, ¿qué diferencia hay entre un componente y una clase? La respuesta es simple. Todos los componentes son clases. Lo que hace que los componentes software sean especiales es que deben cumplir una especificación sobre los mismos. La especificación sobre *JavaBeans* es el documento que describe cuándo una clase Java se debe considerar como un *bean* o componente software desarrollado usando Java.

¿QUÉ ES UN JAVABEAN?

Las definiciones de *JavaBean* son de las más variadas, pero todas tienen en común el hecho de reconocerlos como componentes software reutilizables e intercambiables. Se puede definir un *JavaBean* como un conjunto de una o más clases Java agrupadas en un único fichero tipo *JAR* (*Java Archive*). También puede definirse como un componente aplicado en la construcción de interfaces gráficas de usuario, como el clásico control de Visual Basic.

En cualquier caso, la denominación más simple de *JavaBean* lo muestra como una clase pública Java que implementa un constructor nulo, esto es, sin parámetros. Normalmente dicho constructor consta de unas propiedades, unos métodos y unos eventos que utilizan una serie de convenciones en su nombrado (las que dicta el diseño de patrones).

Los componentes software ofrecen una funcionalidad ocultando su implementación y encapsulando sus datos

De todas formas, como cualquier otro tipo de componente, un *JavaBean* es un conjunto de código que puede ser utilizado y aplicado con un mínimo esfuerzo, garantizando su utilización en los programas de los que forman parte. Entre las ventajas de los *JavaBeans* sobre otros tipos de componentes destacan:

- Son código Java, por lo tanto portable a otras plataformas.
- Pueden ser aplicables en los entornos de desarrollo para su utilización como otro componente más.
- Pueden ser utilizados como ficheros *JAR*, para por ejemplo, implementar *applets*.

UN JAVABEAN SENCILLO

En primer lugar nos vamos a centrar en desarrollar un *bean* sencillo en el que se muestren sus características fundamentales:

```
public class holabean {
      private String cad;

      public holabean () {
        cad = "Hola Mundo";
      }
      public String getMensaje () {
        return cad;
      }
      public void setMensaje ( String literal ) {
        cad = literal;
      }
      public void visuMensaje () {
        System.out.println (cad);
      }
}
```

Como cualquier otro componente software, para ser utilizado se debería ejecutar mediante la siguiente sentencia:

```
new holabean().visuMensaje();
```

Aunque siendo más estrictos, según el diseño de patrones, se tendría que utilizar de la siguiente forma:

```
holabean hb = new holabean();
hb.setMensaje ("mi mensaje");
hb.visuMensaje();
```

Se puede comprobar que el constructor es nulo, no tiene parámetros y se encarga de inicializar las propiedades del *bean*. No existen variables públicas, sino métodos que acceden a esas variables, por lo que se sigue las normas que indican el diseño de patrones y por tanto las de los *beans*.

CARACTERÍSTICAS DE LOS JAVABEANS

La especificación sobre *JavaBeans* determina una serie de características comunes a todos ellos, tales como la instrospection, persistencia, empaquetamiento, interoperación, gestión de eventos y propiedades.

Las propiedades son los atributos de los *beans* y por tanto determinan sus características y comportamiento. Pueden ser desde el color de un botón hasta el tamaño de un contenedor tipo panel. Por ejemplo, *background* es una propiedad que determina el color de fondo de un componente.

Desde un punto de vista de programación, las propiedades son variables miembro de tipo *private* o *protected*, que sólo pueden ser manipuladas a través de determinados métodos, los *accessor method o* métodos de acceso.

La comprobación de la validez de un bean se puede realizar hoy en día mediante asistentes

Los métodos de acceso se suelen clasificar en dos grupos: *setter* y *getter*. El primero es el encargado de asignar un valor a una propiedad y se determina de la siguiente forma:

```
void set<propiedad> ( <tipo> valor );
```

El segundo es aplicado cuando se quiere obtener el valor de una propiedad, siendo su formato el siguiente:

```
void get<propiedad> ( <tipo> valor );
```

También las propiedades se pueden encuadrar en distintos tipos, como son *bound* y *constrained*.

En determinadas circunstancias es útil que cuando se produzca un determinado evento, se modifiquen ciertas propiedades del componente. Las propiedades de tipo *bound* generan eventos que son enviados a otros objetos cuando el valor de una deter-

minada propiedad cambia, permitiendo así al objeto que recibe ese evento realizar algún tipo de acción. En otras ocasiones, ciertos valores de una propiedad pueden ser no válidos para un componente, ya que por ejemplo dependa de estado de otro *bean*. Debido a esto, un *bean* puede discriminar qué tipo de evento le es válido. Este tipo de propiedad se denomina como: *constrained*.

Una propiedad destacable de los *beans* es que no tienen que tener siempre propiedades "visuales". Un *bean* visible es aquel que deriva de *java.awt.Component* y puede ser añadido a contenedores y así ser visualizado. Los *beans* no visibles son aquellos que no se derivan de la clase *component*, sino de *Object* y son *beans* perfectamente válidos.

Finalmente, cabe destacar que todas las clases que permiten la gestión de las propiedades y características de un *bean* se encuentran en el paquete *java.beans*.

INTROSPECTION

El proceso de Introspection permite a los entornos visuales analizar un *bean* y así determinar cómo funciona, que métodos dispone, etc. Este proceso a veces también se denomina *discovery*. Los entornos de desarrollo descubren las características de un *bean* principalmente de dos maneras:

- Preguntando al propio *bean* sobre su descripción y/o las propiedades que posee.
- Analizándolo mediante unos métodos que inspeccionan su construcción.

El primer método se basa en preguntar al *bean* sobre un objeto de tipo *BeanInfo* que lo constituye, el cual describe principalmente sus propiedades. En todo este proceso se buscarán los

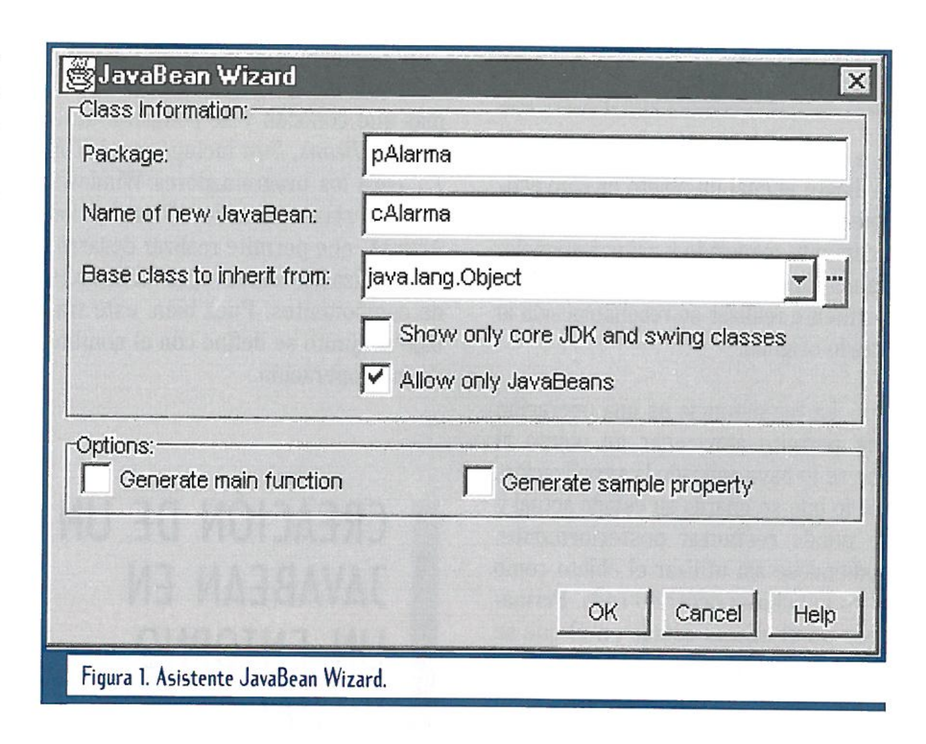


Figura 1. Asistente JavaBean Wizard.

métodos de tipo *setter* y *getter*. Esto significa que si el *bean* sigue las especificaciones sobre JavaBeans los entornos visuales determinarán automáticamente sus características.

El mecanismo de análisis del *bean* es algo más complejo y se basa principalmente en el uso de los métodos descritos en el paquete *java.lang.reflect*.

GESTIÓN DE EVENTOS

Un evento es la ejecución de una determinada acción, lo que provoca la notificación de ese suceso a una serie de objetos. Todas los eventos que ocurren cuando se utilizan los diferentes *beans* de una aplicación presuponen la existencia de un mecanismo que organice la gestión de los eventos producidos. De ello se encarga el nuevo modelo de eventos del *JDK 1.1*.

En el nuevo modelo de eventos, un *bean* puede generar un evento. Cada tipo de evento es representado por una clase diferente. Cuando se produce el evento, éste puede ser aplicable a una o más clases *listener*. Una clase *listener* "registra" todas las acciones que suceden en el componente mediante la utilización de una interfaz denominada *listener interface* y actúa en función del evento.

La portabilidad, su aplicación en entornos visuales y su uso como applets hacen que los JavaBeans sean componentes muy prácticos

El proceso de registro o determinación del evento se realiza mediante la llamada a unos métodos que se definen *addXXXListener()*, donde XXX representa el objeto sobre el que se aplica.

Se puede decir que es un sistema que se basa en que un: "Veamos que ocurre si sucede esto...", determinando la acción que ocurre y sobre qué objeto se aplica. A continuación se genera un evento que es gestionado por el objeto que lo provoca, el cual es siempre un objeto de tipo *EventObject*. Gracias a este tipo, se puede derivar otras clases para generar eventos propios.

PERSISTENCIA Y EMPAQUETAMIENTO

La serialización es un proceso mediante el cual un objeto es convertido en un *stream* de *bytes* que puede ser fácilmente manejado y así por ejemplo, transmitirlo a través de la red y posteriormente realizar su reconstrucción al estado original.

La persistencia es una operación que permite almacenar un objeto al que se le haya aplicado la serialización, por lo que se guarda su estado actual y se puede restaurar posteriormente, pudiéndose así utilizar el objeto como si no le hubiera ocurrido nada. Permanece en el mismo estado en el que se aplicó la persistencia. La utilización de la persistencia por los entornos visuales se basa en almacenar las propiedades de un *bean* para más tarde actualizar su estado y obtener sus propiedades intactas.

La especificación sobre *JavaBeans* describe que un *bean* se construirá a partir de un fichero tipo *JAR* con una estructura de fichero tipo *ZIP*. Un fichero *JAR* puede incluir objetos serializados, documentos, imágenes, etc. Pues bien, el proceso de crear un fichero *JAR* se denomina empaquetamiento. Gracias a este proceso, un *bean* está comprimido y la tarea de descomprimirlo es más eficaz que si un *bean* estuviera formado por múltiples archivos comprimidos, además de que se acelera su transmisión por la red.

INTEROPERACIÓN

Alguien dijo una vez que lo interesante de los estándares es que hay muchos donde elegir. Los componentes software no son precisamente una excepción. Los hay de los más variados en los más distintos sistemas, desde el famoso control *OLE* (*OCX*), las *VBX* de Visual Basic, hasta los *ActiveX* (todos ellos aplicables en los entornos Windows), pasando por los componentes de *OpenDoc* y de *LiveConnect*.

Como no es previsible que los desarrolladores abandonen los sistemas que conocen y se pongan a utilizar *JavaBeans*, *Sun* incluyó en el *JDK 1.1* para los programadores Windows una interfaz entre los *Beans* y los *ActiveX*, que permite realizar desarrollos utilizando estos tipos diferentes de componentes. Pues bien, este trabajo conjunto se define con el nombre de interoperación.

CREACIÓN DE UN JAVABEAN EN UN ENTORNO VISUAL

La utilización de entornos visuales permite la creación de *beans* de una manera fácil y rápida, por lo que la proliferación de estas herramientas ha sido enorme desde la aparición de los *beans* en la versión 1.1 de Java. Para desarrollar un *bean* vamos a utilizar *Jbuilder 2 de Borland*, debido a la gran cantidad de asistentes que disponible para todos los procesos necesarios.

CREAR LA CLASE PRINCIPAL DE UN BEAN

Mediante el asistente BeansExpress podremos crear el *bean* siguiendo los pasos que describimos en las ssiguientes líneas.

En primer lugar seleccionamos la opción **New** dentro de la opción **File** del menú principal. Aparecerá una ventana en la que podremos elegir el asistente *JavaBean Wizard*, con se aprecia en la figura 1.

JavaBean Wizard se encarga de pedirnos que especifiquemos el nombre del *bean* que se va a crear y el paquete en el que se incluirán todas las clases generadas.

En la opción que determina desde qué clase se quiere heredar, descubrimos una serie de opciones entre las que deberemos elegir en función del tipo de *bean* que el usuario desee desarrollar.

Si deseamos usar componentes *Swing* señalaremos **com.sun.java.-swing.Jpanel** o en caso de que sea un *bean* no visual, **java.lang.Object**.

Un bean consta de un constructor nulo, unas propiedades, unos métodos de acceso y unos eventos

Supongamos que vamos a crear un *bean* que sea una alarma, es decir, que genere un evento pasado un determinado intervalo de tiempo. De esta forma, se podrá asociar una acción de respuesta a dicho evento. Si se elige un contenedor de *java.awt* se obtiene el siguiente resultado:

```
package pAlarma;
import java.awt.*;
public class cAlarma extends Panel {
 BorderLayout borderLayout1 = new Border-
      Layout();
 public cAlarma() {
  try { jbInit(); }
  catch (Exception ex) { ex.printStackTrace();}
 }
 private void jbInit() throws Exception {
  this.setLayout(borderLayout1);
 }
}
```

Para el diseño de la interfaz se utiliza la entrada *Design* de la barra inferior de pestañas y junto con los componentes de la barra de herramientas diseñamos su aspecto.

LAS PROPIEDADES DEL BEAN

Supongamos que se está construyendo un *bean* del tipo alarma, como se describió anteriormente. Este *bean* deberá implementarse como un *thread* y a su vez tendrá una propiedad que será el tiempo que transcurre hasta que un evento cualquiera sea lanzado.

Los beans no visibles son aquellos que no se derivan de la clase component

Para añadir las propiedades al *bean* se selecciona la pestaña de *bean* del menú inferior de pestañas. En este momento, dentro de la opción *General*, se presentan dos nuevas opciones, que se deberán marcar en la mayoría de los casos que se realice un proceso de serialización. A continuación se elige la opción **Propiedades** y se selecciona **Añadir Propiedad.** Aparecerá un cuadro de diálogo en el que se introducirá un nombre, un tipo de variable y un tipo de propiedad (*bound*, *constraid*, ninguna). Además se debe indicar si se desea para esa propiedad los métodos *setter* y *getter*. En el caso del *bean* alarma añadiríamos una propiedad *timeout*, como un entero y añadiríamos los dos métodos. En cuanto al tipo de propiedad la opción por defecto *none* sería la más apropiada. Si observamos el código generado, tenemos los siguientes nuevos métodos:

```
void readObject(ObjectInputStream ois) throws
      ClassNotFoundException, IOException {
  ois.defaultReadObject();
}
void writeObject(ObjectOutputStream oos)
      throws IOException {
  oos.defaultWriteObject();
}
public void setTimeout(int newTimeout) {
  timeout = newTimeout;
}
public int getTimeout() {
  return timeout;
}
private int timeout;
```

Estos métodos han sido generados al seleccionar la serialización y al añadir la propiedad *timeout*. El proceso descrito anteriomente se realizará de forma análoga para cualquier otro tipo de propiedad que se requiera.

Si se hubiera definido alguna de las propiedades del tipo *bound* o *constrained*, observaríamos que el asistente habría añadido dentro del código unos métodos de tipo *PropertyChangeListener*, encargados de realizar las tareas correspondientes.

CREACIÓN DE LA CLASE BEANINFO

Como se mencionó anteriormente, para averiguar las características de un *bean* se realiza una consulta a la clase *BeanInfo* asociada. Esta clase es opcional, pero es interesante crearla porque determina claramente qué partes del componente son modificables, qué es accesible y cuáles deben permanecer tal cual porque se utilizan para su funcionamiento interno y no requieren modificación alguna.

En la opción **Bean** y pulsando en **BeanInfo** se muestra una pantalla con

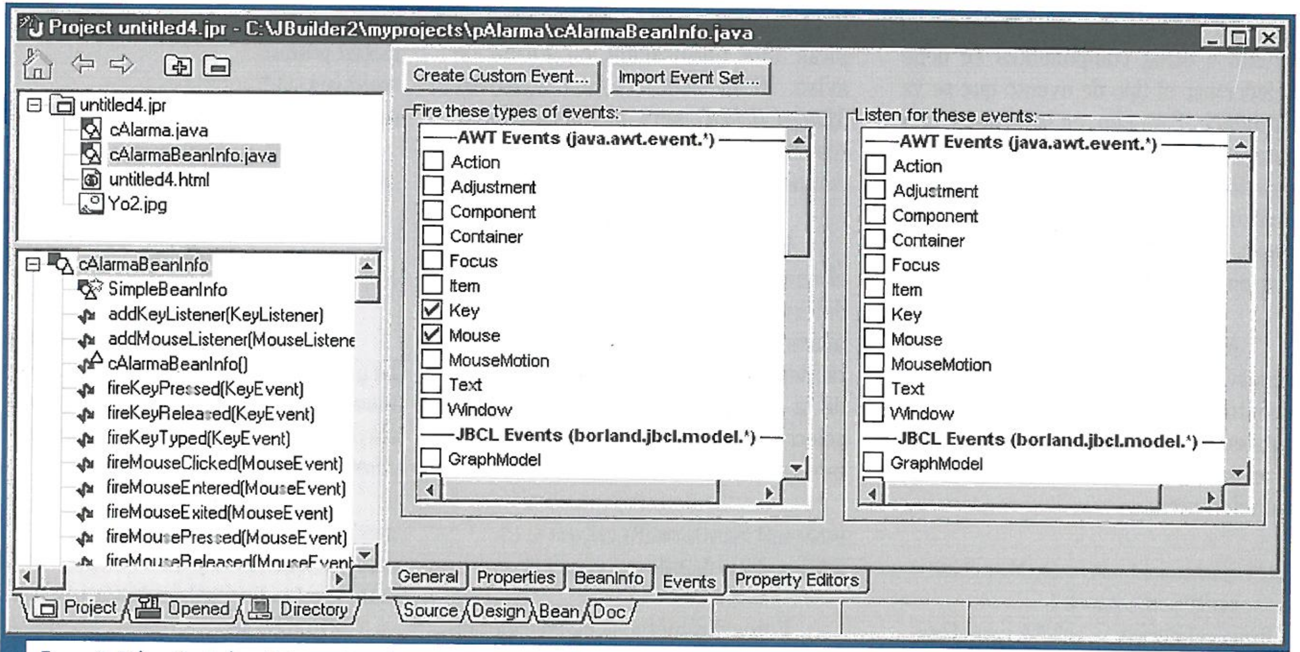


Figura 2. Selección de los distintos tipos de eventos.

todas las propiedades de nuestro *bean*, de las cuales habrá que decidir si aparecerán de cara a la interfaz del componente o no. Para ello únicamente tendremos que marcar la opción de *Expose*. Además en esta opción dentro de la parte **Iconos** se puede especificar el icono gráfico que se desea que aparezca para identificar el *bean*. Una vez hecho esto se pulsa la opción **Generar BeanInfo** y se creara una nueva clase que se denominara como el *bean* creado concatenando la palabra clave *BeanInfo*.

La modificación del *BeanInfo* se realizaría simplemente actualizando las propiedades y volviéndolo a generar.

LOS EVENTOS DEL BEAN

Como se ha mencionado antes, un *bean* puede generar eventos. De esta forma, un objeto que se mantiene a la escucha podrá recibir un objeto de tipo evento. También puede escuchar por nuevos eventos y producir un resultado cuando éstos ocurran.

Para que un *bean* pueda enviar un evento a otros componentes se debe seleccionar el tipo de evento que se va a producir. Para ello, en la opción **Bean** seleccionaremos **Events** y marcaremos los eventos que se deseen lanzar dentro la opción **Fire**, tales como *Key*, *Focus*, *Item*, etc, como se puede apreciar en la figura 2.

Ahora en la parte del código se observa que se ha generado por cada evento tres tipos de funciones como las siguientes, que son las pertenecientes al evento correspondiente a la pulsación del ratón:

```
public synchronized void removeMouseListe-
        ner(MouseListener l) {
  if (mouseListeners != null && mouseListe-
        ners.contains(l)) {
    Vector v = (Vector) mouseListeners.clone();
    v.removeElément(l);
    mouseListeners = v;
  }
}
public synchronized void
        addMouseListener(MouseListener l) {
  Vector v = mouseListeners == null ? new
        Vector(2) : (Vector)
        mouseListeners.clone();
  if (!v.contains(l)) {
    v.addElement(l);
    mouseListeners = v;
  }
}
protected void fireMouseClicked(MouseEvent
        e) {
  if (mouseListeners != null) {
    Vector listeners = mouseListeners;
    int count = listeners.size();
    for (int i = 0; i < count; i++)
      ((MouseListener)
        listeners.elementAt(i)).mouseClicked(e);
  }
}
```

Esto implica que cuando un componente necesita que se le indique que el ratón ha sido pulsado se llamara al método *addMouseListener()* del *bean* y ese componente será añadido como un elemento de la lista *mouseListeners*. De esta forma, cuando se pulse el ratón todos esos componentes añadidos recibirán una notificación en forma de aviso. El método *fire*<evento>(*fire-MouseClicked*) será el encargado de realizar todas las notificaciones a los componentes necesarios.

En el caso de que nuestro *bean* sea el que está esperando a que ocurra un determinado evento, se seguiría el mismo proceso pero los eventos a esperar serían determinados por los eventos de la opción *Listen for these events*. Si seleccionamos la opción *Focus*, obtenemos dos nuevos procedimientos:

```
public void focusGained(FocusEvent e) { }
public void focusLost(FocusEvent e) {}
```

Estos dos métodos están esperando a que un componente que haya registrado el *bean* les envíe un evento para poder ejecutar la tarea que se determine dentro del mismo.

El proceso de Introspection permite descubrir las características de un javabean

También se pueden construir eventos propios para nuestros *javabeans*. Volvamos con el bean genera la alarma. Se podría crear el evento **EventoAlarma** que sería pasado al *listerner* correspondiente, indicando que se ha producido una alarma.

Para crear nuevos eventos mediante un asistente, se selecciona la opción **Bean**, después la pestaña de **Eventos** y a continuación se pulsa el botón que indica **Crear Evento** (*Create Custom Event*). Esta opción podemos observarla en la figura 3.

Como se mencionó, todos los *beans* con eventos propios deben heredar de *java.util.EventObject* o de una subclase de ésta. Por tanto el resultado generado es válido:

```
package palarma;
import java.util.*;
public class EventoAlarmaEvent extends Even-
        tObject {
  public EventoAlarmaEvent(Object source) {
    super(source);
  }
}
```

Además vemos que el asistente ha generado la siguiente interfaz que se encarga de implementar el *listener* a través del cual será notificado el *evento-Alarma*:

```
package palarma;
import java.util.*;
public interface EventoAlarmaListener extends
        EventListener
{
```

```
public void alarmaEjecutar(EventoAlarmaE-
    vent e);
}
```

Todas las *interfaces listeners* de los *JavaBeans* deben ser derivados de *java.util.EventListener* o de una subclase de ésta. En la *interface listener* se pueden declarar todos los métodos que sean necesarios, aunque deberán de tener como parámetro siempre un objeto. El propio *bean se* encargará de determinar que método será ejecutado cuando se produzca un evento. En este caso, el método *alarmaEjecutar* indicará que se ha producido una alarma.

Una vez generados nuestros eventos, éstos aparecen separados del resto por la opción *CustomEvents*.

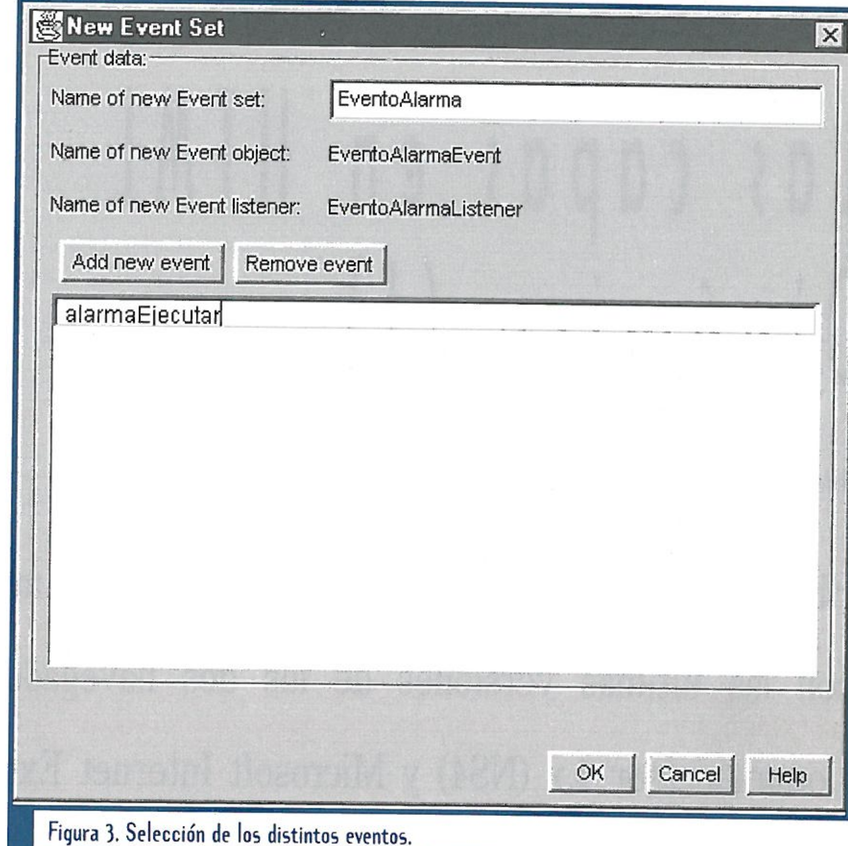


Figura 3. Selección de los distintos eventos.

El empaquetamiento hace que un bean sea un fichero más práctico de gestionar

En este momento la creación del bean ha finalizado. Se compila todo el proyecto mediante la opción **Build** seguida de **Make project** y si todo ha ido bien el *bean* ya se encuentra listo para ser probado

COMPROBAR EL JAVABEAN CREADO

Tras finalizar la creación del *bean* se debe comprobar que sigue las especificaciones indicadas para los mismos. Para ellos utilizaremos el asistente denominado *BeanInsight,* que se encargará de realizar esta tarea. Si *BeanInsight* encuentra algún error lo indica y si no, identifica las propiedades, las características y los eventos del *bean* mediante el proceso de *instropection* de la clase *BeanInfo,* tal y como se indicó anteriormente.

Para comprobar un *bean,* se elige la opción **Wizard** del menú principal y a continuación la opción **BeanInsight**. Se introduce el nombre del *bean* que se desea comprobar y se pulsa el botón que realiza la función de examinar.

En caso de que se produzca algún tipo de error, seleccionando el botón **Ver Detalles** podremos inspeccionar qué tipo de problemas se han encontrado. Típicamente suelen surgir la falta eventos e incluso que no se ha recompilado todo el proyecto entero tras una modificación mínima.

Una vez que el *bean* es válido podemos acabar el proceso de creación empaquetándolo, es decir, generando un fichero JAR. Para ello se usa el asistente utilizando la opción **Deployment Wizard** dentro de **Wizards.** Aparecerá un cuadro de diálogo en el que a la izquierda se muestran todos los elementos del *bean*, las clases, los iconos, la documentación en páginas HTML si se hubiera generado, etc. Se elige generar un fichero *JAR* y se indica el *path* donde crearlo. Por último pulsaremos el botón de instalar y se procederá con la creación del fichero.

CONCLUSIONES

Gracias a los entornos visuales, el desarrollo de aplicaciones que utilizan *beans* es un proceso sumamente fácil y rápido, obteniéndose muy buenos resultados. Así mismo, crear un *bean* es una tarea que prácticamente se puede resolver mediante los múltiples asistentes que existen en los productos *RAD*. En nuestro caso el desarrollo se realizó mediante la JBuilder 2.



Las capas en HTML Dinámico (I)

Adolfo Aladro (aaladro@arrakis.es)

El HTML dinámico ha sido una de las novedades más importantes introducidas con las últimas versiones de los dos navegadores principales: Netscape Communicator 4.x (NS4) y Microsoft Internet Explorer 4.x (IE4). Podríamos asegurar, sin riesgo a equivocarnos, que se trata de una verdadera revolución.

Con el HTML dinámico es posible manipular los elementos de una página web, cambiar su apariencia, o posicionarlos en cualquier lugar dentro de la página. Todo ello sucede dinámicamente en la máquina del usuario sin más software adicional que el propio navegador. En la actualidad existen varias tecnologías que se pueden usar en el desarrollo de soluciones web, entre las que cabe destacar:

- HTML dinámico (DHTML, *Dynamic HTML*)
- Applets
- Controles ActiveX

La posibilidad de poder realizar nuestros sitios web utilizando cualquiera de estas tecnologías produce dudas a la hora de decidirnos por una u otra. Cada tecnología tiene sus puntos fuertes y sus puntos débiles, y debemos escoger teniendo en cuenta las características del trabajo que desarrollar. La Tabla 1 muestra una comparativa entre estas tres tecnologías evidenciando las ventajas y los inconvenientes más importantes de cada una. De la tabla se desprende que el HTML dinámico es una alternativa multiplataforma, de alto rendimiento que ofrece al autor de páginas web un amplio abanico de opciones de creatividad y efectos multimedia.

HTML DINÁMICO

El HTML dinámico es una tecnología basada en estándares gracias a la cual es posible manipular cualquier elemento de una página dinámicamente. Hasta la aparición del HTML dinámico resultaba bastante complicado realizar tareas tales como cambiar textos e imágenes como respuesta a un evento, mover bloques enteros de texto e imágenes desde un punto de partida hasta otro de destino, renovar y actualizar formularios como consecuencia de acciones por parte del usuario, etc.

Las hojas de estilo surgen de la intención de separar la representación de los documentos HTML de su contenido

La única forma que teníamos para llevar a cabo estas tareas era utilizar varias páginas HTML, controles ActiveX, applets Java, o programas que se

Tabla 1. Comparación de tecnologías de desarrollo de soluciones web.

	Ventajas	Inconvenientes
Controles ActiveX	• Integración con herramientas y aplicaciones de Win32. • Reutilizables paran. aplicaciones Windows. • Idóneos para acceder a servicios del sistema. • Pueden ser rápidos, una vez en el cliente, ya que se pueden programar con lenguajes potentes.	• Son sólo totalmente soportados en plataformas Windows. • No se ejecutan en un área protegida. Antes de ejecutarse primero es necesario que el usuario decida si confía o no en el control.
Applets Java	• Java es fácil de aprender si ya se conoce C++ o C. • Se pueden crear avanzados interfaces de usuario para múltiples plataformas. El lenguaje Java proporciona más funcionalidad que HTMl y los lenguajes de script.	• Aunque se pueda, teóricamente, escribir un applet de Java multiplataforma, hay que probarlo en todas las plataformas. • Los applets de Java no son necesariamente pequeños y rápidos.
HTMl Dinámico	• Permite crear interfaces de usuario interactivos. • Ya que está basado en HTMl y lenguajes de script, pueden aprovecharse los conocimientos previamente adquiridos de HTML. • Es multiplataforma. • Es una tecnología abierta basada en estándares. • Es rápido y pequeño. Debido a que es interpretado por el Navegador, no necesita de componentes auxiliares (.dll) para su ejecución, como ocurre con controles ActiveX y applets Java.	• Microsoft y Netscape tienen diferentes implementaciones de DHTML. • Carece de la potencia de los controles ActiveX o los appelts Java, por lo que no es adecuado para aplicaciones complejas.

ejecutaran en el servidor. El problema que presentaban estas soluciones era que en general implicaban un menor rendimiento, y además añadían complejidad a los proyectos al tener que tratar con varias tecnologías distintas a la vez.

Con el HTML dinámico es posible manipular los elementos de una página web, cambiar su apariencia, o posicionarlos en cualquier lugar dentro de la página

Hoy muchos de estos efectos pueden conseguirse con HTML dinámico y cada vez son más los ejemplos que encontrarnos en Internet: juegos interactivos (ver figura 1), menús desplegables (ver figura 2), presentaciones multimedia, etc.

El HTML dinámico es el resultado del avance de las siguientes tecnologías web:

- **HTML 4.0**
  El lenguaje HTML es el núcleo central de esta tecnología. Con cada versión el número de etiquetas y la potencia de las mismas ha ido aumentando, dotando así al desarrollador de páginas web de más posibilidades de diseño y creatividad.

- **CSS (Cascading Style Sheet)**
  Las hojas de estilo surgen de la intención de separar la presentación de los documentos HTML de su contenido. Se trata pues de crear estilos que luego podamos aplicar a determinadas etiquetas HTML para que los elementos definidos por ellas se visualicen con el estilo correpondiente.

- **DOM (Document Object Model)**
  El *DOM* informa al usuario sobre los elementos y atributos de la página que pueden manipularse a través de lenguajes de *script*, así como de la jerarquía de objetos que estos elementos constituyen (ver figura 3). La evolución que ha sufrido el *DOM* ha permitido que cada vez sea mayor el número de elementos y atributos susceptibles de ser modificados dinámicamente utilizando lenguajes de script.

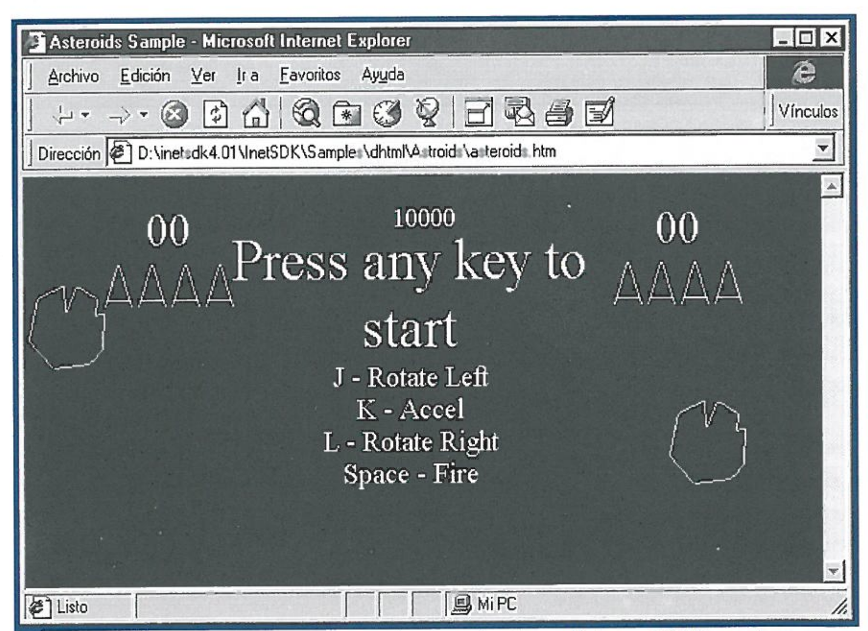


Figura 1. El tradicional juego interactivo de asteroides desarrollado con DHTML para Internet Explorer 4.0.

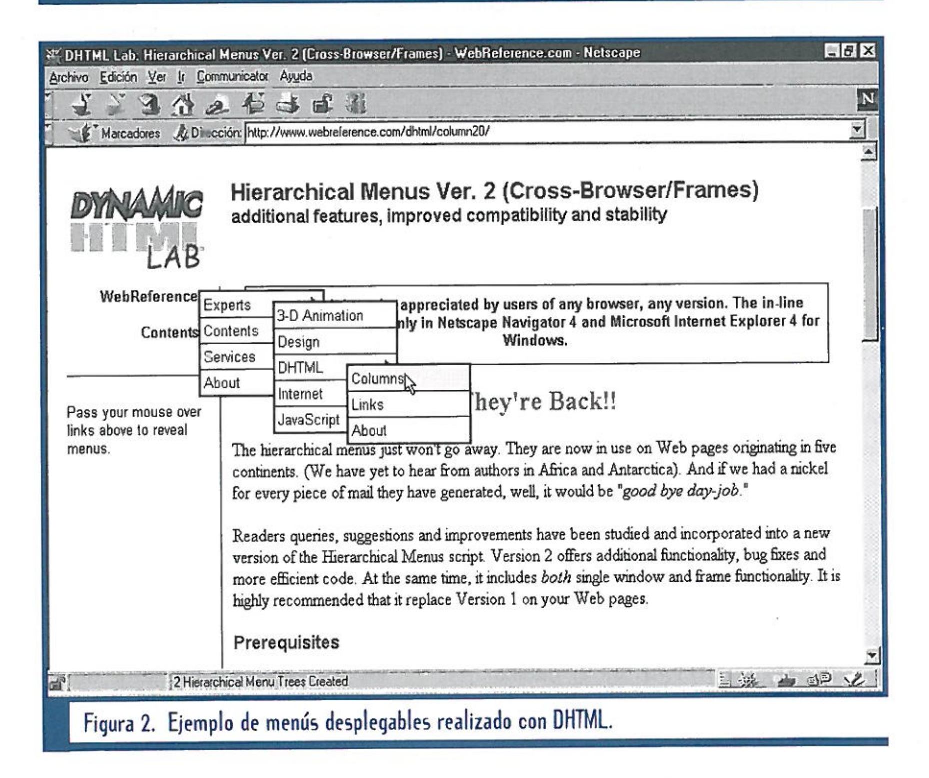


Figura 2. Ejemplo de menús desplegables realizado con DHTML.

- **Lenguajes de script (VBScript, JScript o JavaScript)**
  Desde su nacimiento los lenguajes de script han ido pasando por varias versiones. Cada una ha venido a aportar mayor capacidad de control dinámico de los elementos contenidos en un documento HTML, al mismo tiempo que más recursos.

Este artículo y otros futuros pretenden ser una aproximación fundamentalmente práctica al HTML dinámico compatible. El lector habrá notado que hemos introducido un adjetivo que apenas si lo habíamos usado de pasada hasta ahora: compatible. Efectivamente uno de los inconvenientes más importantes con los que cuenta esta tecnología es el hecho de que IE4 y NS4 tienen formas, a veces ligeramente distintas, otras radicalmente opuestas, de implementar el HTML dinámico.

Las hojas de estilo no son más que una definición de un conjunto de atributos que controlan la presentación de los elementos de una página web

Cualquier autor de páginas web medianamente experimentado sabrá que en la mayoría de los casos se le solicitará que los sitios web que cree funcionen tanto para IE4 como para NS4, e inmediatamente se dará cuenta de que mantener dos versiones de las mismas páginas es una solución cuando menos engorrosa. Afortunadamente existe un subconjunto de características comunes a los dos navegadores que hacen posible, con un poco de ingenio, la realización de sitios web compatibles. Tomando como punto de partida ese subconjunto común trataremos de sacarle el mayor provecho posible al HTML dinámico.

LAS HOJAS DE ESTILO

Las hojas de estilo no son más que una definición de un conjunto de atributos que controlan la presentación de los elementos de una página web. Una vez que hemos creado un estilo podemos aplicarlo a una etiqueta

Tabla 2. Resumen de las propiedades más importantes de las hojas de estilo.

Propiedades de la fuente	
font-family	Fuente del sistema. Por ejemplo: {font-family: verdana}.
font-weight	Intensidad. Puede ser: normal, bold, bolder o lighter.
font-size	Tamaño de la fuente. Se expresa con valores numéricos: puntos (pt), pulgadas (in), centímetros (cm) o pixels (px). Por ejemplo: {font-size: 10pt}.
font	Modo general de asignación de cualquiera de las anteriores.
Propiedades del color	
color	Color del texto. Puede expresarse en valores RGB (p.ej.: #AA33BB), o mediante mnemónicos (p.ej.: red). Por ejemplo: {color: black}.
background	Color del fondo. Se expresa igual que el anterior.
Propiedades del texto	
text-decoration	Efecto del texto. Puede ser none, underline o line-through. Por ejemplo: {text-decoration: none}.
text-align	Alineamiento. Puede ser: left, right, center o justify. Por ejemplo: {text-align: left}.
text-indent	Indentado. Se expresa con valores numéricos. Por ejemplo: {text-indent: 10px}.
Propiedades de bloque	
margin-right	Margen derecho del bloque. Se expresa con valores numéricos. Por ejemplo: {margin: 10px}.
margin-left	Margen izquierdo del bloque. Se expresa igual que el anterior.
margin-top	Margen superior. Se expresa igual que el anterior.
margin-bottom	Margen inferior. Se expresa igual que el anterior.
margin	Modo general de asignación de cualquiera de las anteriores.
border-style	Estilo del borde. Puede ser: none, dotted, dashed, solid, double o groove, ridge, inset o outset.
border-color	Color del borde. Se expresa igual que color.
border-right-width	Ancho del borde derecho del bloque. Se expresa mediante mnemónicos (thin, medium o thick) o con con valores numéricos. Por ejemplo: {border-right-width: thin}.
border-right	Modo general de asignación de border-right-width, border-style y border-color. Por ejemplo: {border-right: thin dotted #800080}.
border-left-width	Ancho del borde izquierdo. Se expresa igual que border-right-width.
border-left	Modo general de asignación de border-left-width, border-style y border-color. Se expresa igual que border-right.
border-top-width	Ancho del borde superior. Se expresa igual que Igual que border-right-width.
border-top	Modo general de asignación de border-top-width, border-style y border-color. Se expresa igual que border-right.
border-bottom-width	Ancho del borde inferior. Se expresa igual que Igual que border-right-width.
border-bottom	Modo general de asignación de border-bottom-width, border-style y border-color. Se expresa igual que border-right.
border	Modo general de asignación para todos de los bordes
padding-right	Distancia entre el borde derecho y el contenido del bloque. Se expresa con valores numéricos. Por ejemplo: { padding-right: 10px}.
padding-left	Distancia entre el borde derecho y el contenido del bloque. Se expresa igual que el anterior.
padding-top	Distancia entre el borde derecho y el contenido del bloque. Se expresa igual que el anterior.
padding-bottom	Distancia entre el borde derecho y el contenido del bloque. Se expresa igual que el anterior.
padding	Modo general de asignación de cualquiera de las anteriores.

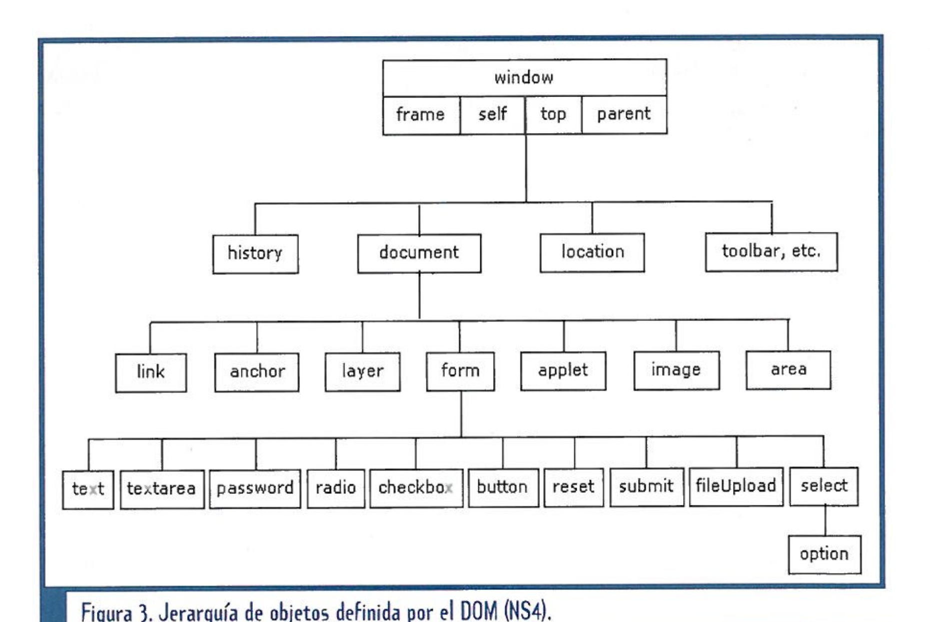


Figura 3. Jerarquía de objetos definida por el DOM (NS4).

HTML y esta se visualizará siguiendo el estilo. Así por ejemplo:

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
.miparrafo {
 font-family: Verdana;
 font-size: 10pt;
}
</STYLE>
</HEAD>
<BODY>
...
<P CLASS="miparrafo">
...
</P>
</BODY>
</HTML>
```

Hará que el párrafo definido por la etiqueta P se visualice con letra del tipo de *Verdana* y de tamaño *10pt*. Las propiedades de las hojas de estilo representan los atributos que podemos cambiar. La Tabla 2 muestra un resumen de las principales propiedades.

Tal y como ya hemos apuntado *NS4* e *IE4* no sólo no implementan fielmente el estándar propuesto por el W3C (*http://http://www.w3.org/ pub/ WWW/TR/REC-CSS1*) sino que además presentan comportamientos en muchos casos opuestos o añaden propiedades exclusivas. De ahí que, si queremos realizar proyectos que sirvan para ambos navegadores, las hojas de estilo han de utilizarse con sumo cuidado. A lo largo de este y otros futuros artículos iremos sacando a la luz algunas de estas incompatibilidades y veremos formas de soslayarlas.

CÓMO DEFINIMOS UNA CAPA

Una capa es un bloque que tiene identidad propia. En HTML podemos crear capas utilizando dos etiquetas: *DIV* y *SPAN* (No debemos olvidar que también es posible crear capas de forma dinámica mediante un script, pero eso de momento vamos a dejarlo de lado). Para el tema que nos concierne basta saber que la diferencia principal existente entre las etiquetas *DIV* y *SPAN* es que mientras la primera implica un salto de línea la segunda no (Comparar la figura 4 con la figura 5).

Una capa queda definida por un identificador único que la distingue de todas la demás, y por un estilo que determina sus características de presentación (posicionamiento, fuente y tamaño de la letra, color de fondo, etc.). Existen varias formas de establecer el identificador así como el estilo, y todas ellas juegan con tres atributos que son comunes a las etiquetas *SPAN* y *DIV*. Estos atributos son: *ID*, *CLASS* y *STYLE*.

- **ID:** Indica el identificador de la capa.
- **CLASS:** Indica la hoja de estilo que contiene el estilo de la capa.
- **STYLE:** Indica el estilo de la capa.

La Tabla 3 muestra las tres formas que tenemos de crear una capa.

En el caso A definimos un estilo mediante la etiqueta *STYLE*. Este estilo estará unívocamente vinculado a una capa, es decir, el nombre del estilo será al mismo tiempo el identificador de la capa y en la declaración dentro de la etiqueta *STYLE* estará precedido del símbolo almohadilla '#'.

Las propiedades de las hojas de estilo representan los atributos que podemos cambiar

En el caso B lo que hacemos es definir un estilo genérico que luego podemos aplicar a tantas capas como queramos. En la declaración dentro de la etiqueta *STYLE* estará precedido del símbolo punto ('.'). Para distinguir unas capas de otras hacemos uso del atributo *ID*.

Finalmente en el caso C introducimos el estilo de la capa mediante el atributo *STYLE* y, como ocurría en el caso anterior, es necesaria la presencia del atributo *ID*, el cuál nos permitirá diferenciar una capa de otra.

POSICIONAMIENTO DE ELEMENTOS MEDIANTE ESTILOS

Otra de las particularidades introducidas por las hojas de estilo es el posicionamiento de elementos (Podemos consultar el estándar propuesto por el W3C en *http://www.w3.org/TR/WD-positioning-19970819*).

En HTML podemos crear capas utilizando dos etiquetas: DIV y SPAN

Desde el punto de vista de las hojas de estilo, un documento HTML es una plano cuyo origen de coordenadas se sitúa en la esquina superior izquierda. Así pues, podemos colocar cualquier elemento dentro de la página en la posición que deseemos. Además también tenemos la facultad para ocultar o mostrar elementos, así como un mecanismo para controlar la forma en la que las capas se apilan, es decir, qué capas tapan a qué otras capas cuando éstas se superponen. Veamos a continuación cuáles son las propiedades que controlan todo esto.

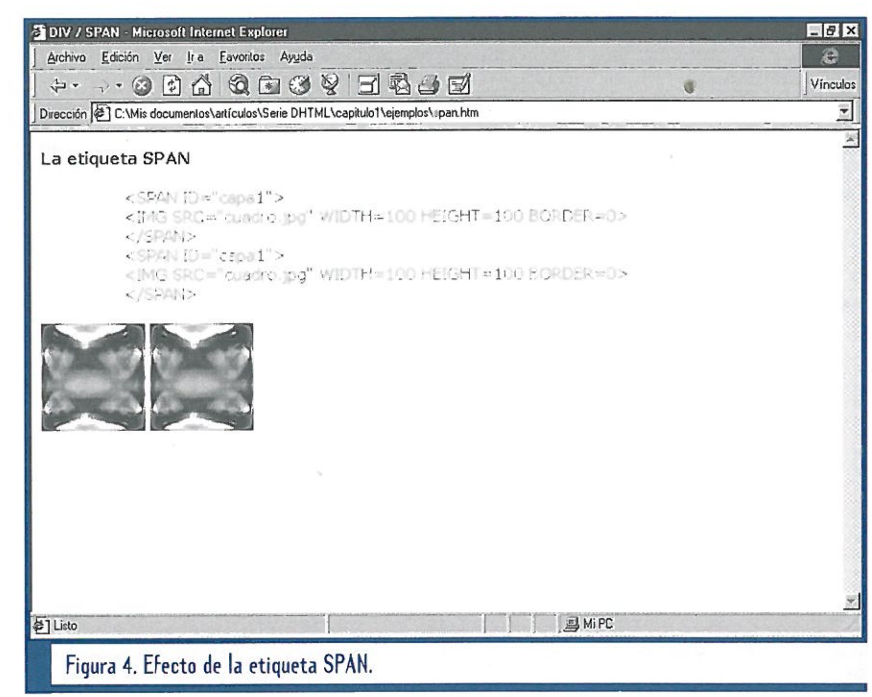


Figura 4. Efecto de la etiqueta SPAN.

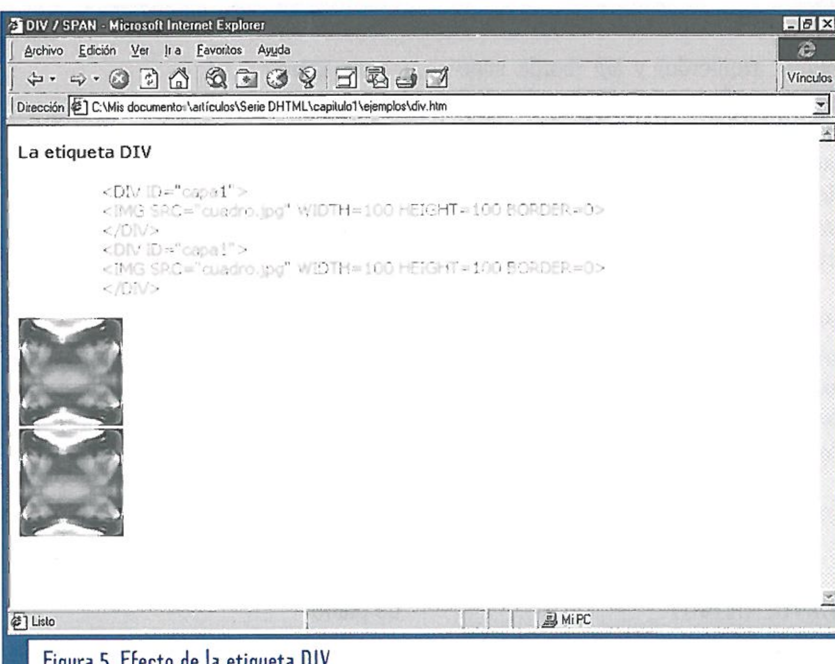


Figura 5. Efecto de la etiqueta DIV.

POSITION

Existen dos tipos de posicionamiento: absoluto (se indicará como *position:absolute*) y relativo (se indicará como *position: relative*). Si no se especifica nada el valor por defecto será *relative*. El posicionamiento absoluto implica que la capa se situará en una posición determinada dada por sus coordenadas, tomando como origen de referencias -el punto (0,0)- la esquina superior izquierda del elemento que contenga a la capa (Normalmente será el documento HTML, pero también puede darse la circunstancia de capas anidadas).

El posicionamiento relativo da como resultado el que la capa aparezca inmediatamente después del elemento HTML que la precede en el código fuente de la página. Es decir, tal y como se posicionan normalmente los elementos (formularios, imágenes, texto, tablas, etc.) en una página: uno detrás de otro a medida que van escribiéndose.

LEFT, TOP

En el caso de que el posicionamiento sea absoluto, las propiedades *left*

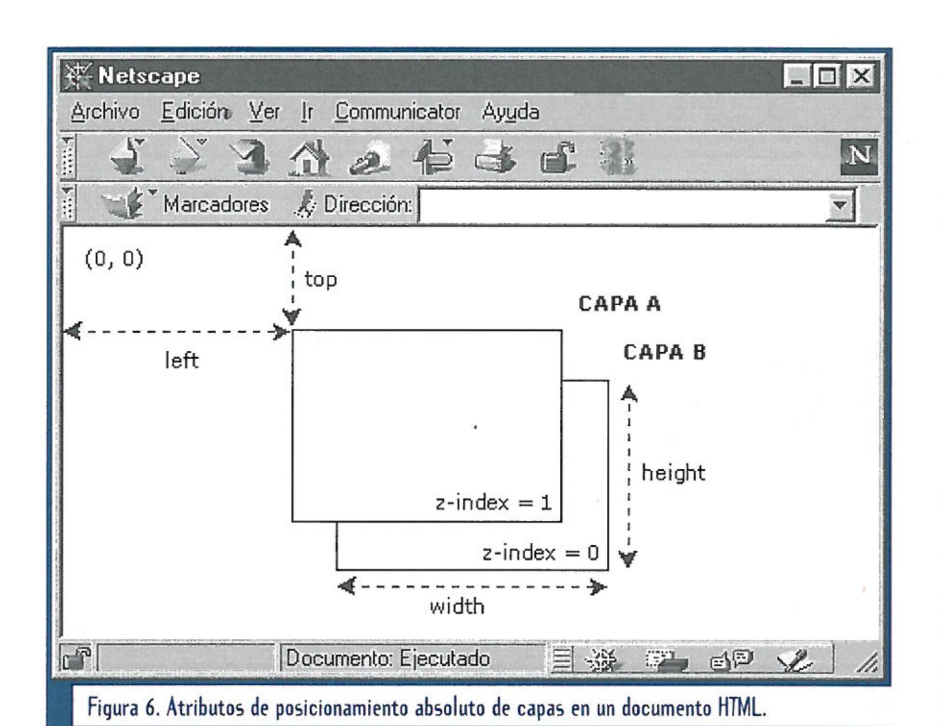


Figura 6. Atributos de posicionamiento absoluto de capas en un documento HTML.

(border izquierdo) y *top* (borde superior) nos proporcionan la situación de la capa con respecto al origen de coordenadas (la esquina superior izquierda del elemento que contenga a la capa). Normalmente estos valores se expresan en *pixels* (px).

```
<STYLE TYPE="text/css">
.miestilo {
  position: absolute;
  left: 100px;
  top: 100px;
}
</STYLE>
```

WIDTH, HEIGHT

Las propiedades *width* y *heigth* determinan respectivamente el ancho y el alto de la capa. De la misma forma, tal y como hemos visto en el caso anterior, sus valores suelen expresarse en *pixels* (px).

```
<STYLE TYPE="text/css">
.miestilo {
  position: absolute;
  left: 100px;
  top: 100px;
  width: 320px;
  height: 200px;
}
</STYLE>
```

Z-ORDER

En un documento HTML pueden existir tantas capas como queramos. Éstas pueden estar distantes o pueden solaparse. En este caso, la propiedad *z-index* determinará el orden de apilamiento de las capas de manera que la capa que se sitúe en primer plano ocultará el trozo que tape de la capa que aparezca en segundo plano.

La primera capa será aquella que tenga el valor de *z-index* mayor. La propiedad *z-index* puede ser *auto* (se asigna automáticamente) o puede tener un valor numérico.

En el caso de que sea *auto*, la propiedad *z-index* de la primera capa que aparezca en el documento tomará un valor igual a 0, y las siguientes 1, 2, 3, etc., de lo que resulta que la última capa será la que esté en primer plano.

VISIBILITY

Esta propiedad determinará si la capa es visible (*visibility:visible*) , o por el contrario, está oculta (*visibility:hidden*).

La figura 6 muestra un ejemplo de las propiedades de posicionamiento.

NS4 VS. IE4

Evidentemente lo primero que tenemos que realizar a la hora de desarrollar un *script* compatible es detectar el tipo y la versión del navegador con el que estamos visualizando la página. Para ello el lenguaje *JavaScript* proporciona el objeto *navigator*.

La propiedad appname nos informará acerca del tipo de navegador

La propiedad *appVersion* de este objeto nos informará de la versión del mismo, y la propiedad *appName* lo hará acerca del tipo de navegador. A lo largo de nuestro *script* serán muchas las veces en las que haremos una cosa u otra dependiendo de estos valores por lo que lo más práctico resulta tener guardado desde el principio esta información en alguna variable. Así podemos hacer:

```
var NS4 = ((parseInt(navigator.appVersion) >=
    4)&&(navigator.appName.indexOf("Nets-
    cape") != -1));

var IE4 = ((parseInt(navigator.appVersion) >=
    4)&&(navigator.appName.indexOf("Explo
    rer") != -1));
```

Las variables *NS4* y *IE4* tendrán **true** o **false** dependiendo de las características del navegador y podremos utilizarlas en cualquier parte de nues-

tro *script* con el fin de realizar una u otra acción.

IE4 y NS4 tienen distintas implementaciones del *DOM*, lo que se traduce en que debemos acceder desde nuestros *scripts* de forma diferente a los elementos de la página. Supongamos que tenemos una capa definida de la siguiente forma:

```
<DIV ID="micapa" CLASS="miclase">
...
</DIV>
```

Con NS4 podemos acceder a las propiedades de esa capa haciendo:

```
document[mi_capa].propiedad
```

Mientras que en IE4 primero tenemos que acudir al objeto *all*, que contiene todos los elementos de la página, y después al objeto *style*, que es el que proporciona todas las propiedades del estilo de la capa:

```
document.all[mi_capa].style.propiedad
```

El posicionamiento absoluto implica que la capa se situará en una posición determinada dada por sus coordenadas

Teniendo en cuenta lo anterior podemos optar por varias alternativas. La más evidente consiste sencillamente en escribir sentencias condicionales (*if-then-else*) cada vez que necesitemos acceder a las propiedades de una capa. Esta solución suele ser bastante engorrosa ya que nos vemos obligados a "sembrar" de *if-then-else* nuestro *script*, lo que dificulta gravemente la claridad del mismo conforme el número de líneas de código aumenta.

Sería mucho más práctico tratar de hallar una forma general de acceder a una capa, que fuera independiente del navegador. El lenguaje *JavaScript* proporciona la función *eval*, la cual evalúa una expresión *JavaScript* dada en forma de cadena de texto. Pues bien, si creamos las siguiente varibles:

```
var layerObj = (NS4) ? 'document' : 'document.all';
var styleObj = (NS4) ? '' : '.style';
```

La llamada a *eval*:

```
eval(layerObj + '["' + nombreCapa + '"]' + styleObj);
```

Nos devuelve una referencia a la capa, tanto en NS4 como en IE4. Por ejemplo:

```
var obj = eval(layerObj + '["' + nombreCapa + '"]' + styleObj);

alert(objt.left);
```

El código anterior funcionaría tanto en NS4 como en IE4. La varible *obj* contendría una referencia a la capa independiente del navegador.

Pero aún podemos ir muchos allá en este modelo de abstracción. Si nos damos cuenta, en realidad lo que estamos tratando de buscar es un objeto capa universal, que podamos manejar en *scripts* hechos tanto para NS4 como para IE4.

Una capa queda definida por un identificador único que la distingue de todas la demás

Una vez creado ese objeto, las diferencias entre los dos navegadores quedarían ocultas en la propia lógica del objeto, y nosotros simplemente

Tabla 3. Diferentes formas de crear una capa dentro de un documento HTML.

Caso A

```
<STYLE TYPE="text/css">
#micapa {
  position: absolute;
  font-family: Verdana;
  font-size: 11pt;
}                          <DIV ID="micapa">
...
</DIV>
```

Caso B

```
<STYLE TYPE="text/css">
.miclse {
  position: absolute;
  font-family: Verdana;
  font-size: 11pt;
}                          <DIV ID="micapa" CLASS="miclase">
...
</DIV>
```

Caso C

```
<DIV ID="micapa" STYLE="position: absolute; font-family: Verdana; font-size: 11pt;">
...
</DIV>
```

Listado 1. Contructor del objeto objetoCapa.

```
function objetoCapa(nombreCapa) {
  objeto = eval(layerObj + '["' + nombreCapa + '"]' + styleObj);
  objeto.nombre = nombreCapa;
  return objeto;
}
```

llamaríamos a las propiedades y los métodos de la misma forma en cualquier navegador. El lector se habrá dado cuenta inmediatamente de que estamos hablando de un modelo de programación orientado a objetos. Efectivamente el lenguaje *JavaScript* también ofrece la posibilidad de definir nuestros propios objetos. El Listado 1 muestra la función constructor que se encarga de crear un nuevo objeto del tipo **objetoCapa**.

ANIMACIÓN

Ahora que sabemos como crear y posicionar una capa dentro de un documento HTML podemos pasar a estudiar el movimiento continuo desde un punto origen hasta otro final, es decir, la animación de una capa.

En un documento HTML pueden existir tantas capas como queramos

Existen muchas formas de crear animaciones, y es posible que cada caso particular tenga sus propias características que lo distingan de los demás, pero nosotros vamos a tratar de buscar una solución general que pueda servir de base para cualquier tipo de animación. Como ya hemos dicho, animar una capa es tan sencillo como moverla de un punto a otro de forma continua. Por lo tanto el primer concepto con el que nos tropezamos es el de ruta, que no es más que el conjunto de puntos por donde pasará la capa en su movimiento desde el origen hasta el destino. La ruta de una animación puede generase en tiempo real. Por ejemplo, tomamos los puntos origen y destino, y calculamos la ecuación de la recta que los une:

Punto origen (x_0,y_0)

Punto destino (x_1,y_1)

$$\frac{x-x_0}{x_0-x_1}=\frac{y-y_0}{y_0-y_1}$$

$$y=\left(\frac{y_0-y_1}{x_0-x_1}\right)\cdot(x-x_0)$$

Aplicando esa ecuación y haciendo variar el valor de x desde $x = x_0$ hasta $x = x_1$ obtenemos todos los puntos por donde ha de pasar la capa que animamos. Este método presenta como mayor inconveniente el hecho de que calcular en tiempo real los puntos introduce un factor de retardo en la animación. Si este factor es considerable, dependiendo de la máquina en donde se esté visualizando la página, pueden producirse trompicones o saltos a lo largo del movimiento. Por eso, una solución mucho más eficiente consiste en tener previamente calculada la ruta de puntos.

De esta manera no se pierde tiempo en hallar cuál va a ser el punto siguiente y la animación es mucho más fluida. Así vamos a definir en *JavaScript* la ruta como una cadena (*String*) de números separados por coma.

Origen				Destino
(x1,y1)	(x2,y2) ···	(xi,yi)	···	(xn,yn)

En JavaScript:

```
ruta = "x1,y1,x2,y2,...,xi,yi,...,xn,yn"
```

Además de todo lo que hemos visto hasta ahora con respecto a la ruta, también debemos contemplar la posibilidad de hacer bucles. Es decir, una animación puede ser finita: vamos de un punto origen a otro destino y paramos. Pero también se da el caso en el que la animación no tiene fin: vamos del origen al destino y volvemos al origen de nuevo repitiendo así todo el recorrido en un bucle infinito. La variable que va a determinar de qué tipo de animación se trata la vamos a denominar *bucle*: será *true* si se trata de una animación infinita y *false* en otro caso.

El lenguaje JavaScript ofrece la posibilidad de definir nuestros propios objetos

Finalmente, el último agente que tener en cuenta es la velocidad de la animación. Como ya veremos más adelante existen dos factores que determinan la velocidad: uno es inherente a la forma en la que se calcula la ruta y otro viene dado por la propia forma de realizar la animación en *JavaScript*.

Por el momento vamos a hablar del primero. Supongamos que estamos calculando la ruta en tiempo real tal y como hemos explicado. El proceso podría ser en *JavaScript* algo parecido a esto:

```
for(x=x0; x<=xn; x++) {
  y = ((y0-y1)/(x0-x1))*(x-x0);
}
```

Pero, ¿por qué ir incrementando el valor de x en un unidad y no en dos, tres o cualquier otro valor?

Evidentemente cuanto mayor sea incremento, menos puntos recorrerá la capa entre el origen y el destino, y por lo tanto el movimiento concluirá antes. Claro que también es cierto que la animación es menos fluida a medida que el número de puntos del recorrido decrece. En cualquier caso este es un parámetro que debemos manejar. Será necesario probar con distintos valores a la hora de calcular la ruta (tanto si lo hacemos en una etapa previa como si lo hacemos en tiempo real).

Bien, hasta aquí digamos que hemos visto el trasfondo más o menos teórico de una animación. Ahora nos queda ver cómo lo implementamos en nuestro *script*. Para empezar vamos a añadir a nuestro objeto capa un método que sirva para inicializar los parámetros de la animación (ver Listado 2).

La ruta de una animación puede generarse en tiempo real

El método *animacion* toma como parámetros de entrada la ruta, que ya hemos dicho que es una cadena formada por números separados por comas; un valor booleano que indica si la animación será finita o no; y un tercer valor que corresponde a la velocidad de la animación. Con estos parámetros asigna al objeto capa una serie de propiedades nuevas así como dos métodos:

RUTA

El método *split* del objeto *String* en *JavaScript* divide la cadena en *tokens* teniendo el cuenta una cadena separadora que toma como parámetro, y devuelve un *array* en el que cada elemento tiene uno de los *tokens* obtenidos. Así por ejemplo, si ruta tiene la cadena:

```
ruta = "100,200,101,201,102,202";
```

Listado 2. Método que inicializa los parámetros de la animación asociada a una capa.

```
function animacion(ruta, bucle, velocidad) {
  this.ruta = ruta.split(',');
  this.indice_ruta = 0;
  this.bucle = loop;
  this.velocidad = velocidad;
  this.animar = animar;
  this.fin = fin;
}
```

Y llamamos al método *split* con la cadena separadora ',':

```
array_rokens = ruta.split(',');
```

El *array* contenido en *array_tokens* será:

0	1	2	3	4	5
100	200	101	201	102	202

INDICE_RUTA

Esta propiedad que creamos para el objeto capa contendrá en todo momento el índice que nos determina en qué posición se haya el recorrido. Evidentemente esta propiedad se utilizará para recorrer el *array* obtenido anteriormente mediante el método *split*.

BUCLE

Esta propiedad simplemente guarda el valor booleano que nos informa acerca de si la animación es finita o no.

VELOCIDAD

La primera forma que se nos puede ocurrir para llevar a cabo la animación consiste en recorrer la ruta mediante un bucle *for*. Sin embargo con este método no tenemos ningún control sobre el tiempo que transcurre entre movimientos sucesivos de la capa.

Un bucle for podría ejecutarse demasiado rápido en un ordenador potente y la animación resultaría acelerada

Un bucle *for* podría ejecutarse, por ejemplo, demasiado rápido en un ordenador potente y la animación resultaría acelerada.

Para evitar este tipo de situaciones creamos el método *animar*, el cual se encarga de realizar el siguiente movimiento de la ruta cada vez que se ejecuta. Este método habrá de ejecutarse tantas veces como puntos tenga la ruta (o infinitas veces en el caso de animaciones no finitas). Para ello el lenguaje *JavaScript* proporciona la función *setTimeout* la cual es capaz de evaluar una determinada función pasado un tiempo. Este factor de tiempo es el que vamos a guardar en la propiedad *velocidad*. Es decir, la propiedad *velocidad* contendrá la medida del tiempo que transcurre entre un movimiento de la capa y el siguiente.

ANIMAR

Este será el método encargado de realizar la animación propiamente dicha. El Listado 3 se encarga de mostrarnos el código.

dará una idea global de la gran potencia y utilidad del HTML dinámico.

Listado 3. Método animar.

```
Function animar() {
  if (this.indice_ruta <= (this.ruta.length-2)) {
    this.mover(parseInt(this.ruta[this.indice_ruta]),parseInt(this.ruta[1+this.indice_ruta]));
    this.indice_ruta += 2;
    setTimeout('coleccionObjetosAnimados["'+ this.nombre + '"].animar()', this.velocidad);
  } else {
    if (this.loop) {
      this.indice_ruta = 0;
      setTimeout('coleccionObjetosAnimados["'+ this.nombre + '"].animar()', this.velocidad);
    } else {
      this.indice_ruta = 0;
      this.fin();
    }
  }
}
```

FIN

En el caso de animaciones finitas este método será llamado cuando la animación haya terminado. Nos servirá para sincronizar las acciones que han de realizarse tras la animación.

EL EJEMPLO

En el CD-ROM de la revista se encuentra un ejemplo que muestra cómo podemos aplicar todo lo que hemos visto hasta ahora a lo largo de este artículo.

Debemos cargar la página **00.htm** la cual abre una nueva ventana con las dimensiones adecuadas. La página que será cargada en esa nueva ventana es la correspondiente al archivo **dhtml.htm**.

El código *JavaScript* se distribuye en dos partes: por un lado dentro de la misma página **dhtml.htm** tenemos la lógica que controla todo el proceso y en **dhtml.js** tenemos el código relativo a nuestro objeto capa.

El funcionamiento de este ejemplo es muy sencillo: una vez que se visualiza la ventana que se abre inmediatamente después de cargar la página **00.htm**, basta con hacer clic en cualquier parte del documento para que las capas que contienen las imágenes se muevan.

Cuando el movimiento finaliza se mostrará una tercera capa. Si hacemos clic retornamos al estado inicial.

CONCLUSIÓN

Con este artículo nos hemos dedicado a introducir el HTML dinámico de una manera sencilla a la vez que práctica realizando una primera visión dentro de este terreno.

Aunque sólo hemos empezado a ilustrar algunas de las posibilidades, que nos ofrece este tema, con el contenido de estas páginas ya tenemos material suficiente como para desarrollar efectos visuales sofisticados y atractivos.

A lo largo de los siguientes artículos iremos profundizando en todos estos temas paso a paso y nos dedicaremos a ver otros nuevos, lo que nos

DIRECCIONES DE INTERÉS

Estándares

- Especificación HTML 4.0
  http://www.w3.org/TR/REC-html40/
- Cascading Style Sheets, nivel 1 (CSS1)
  http://www.w3.org/pub/WWW/TR/PR-CSS1
- Especificación DOM (Document Object Model)
  http://www.w3.org/TR/WD-DOM/

Nestscape

- Documentación sobre DHTML
  http://developer.netscape.com/library/documentation/communicator/dynhtml/index.htm
- Recursos acerca de DHTML
  http://developer.netscape.com/library/documentation/htmlguid/dynamic_resources.html
- Ejemplos de DHTML
  http://developer.netscape.com/docs/examples/index.html?content=dynhtml.html

Microsoft

- DHTML – SiteBuilder
  http://www.microsoft.com/sitebuilder/workshop/author/dhtml/
- Documentación acerca de DHTML (Internet Client SDK)
  http://www.microsoft.com/msdn/sdk/inetsdk/help/
- Galería DHTML
  http://www.microsoft.com/gallery/files/html/

Delphi 4: ¿Alguien da más?

Enrique de la Lastra (elastra@redestb.es)

Aparece en el mercado una nueva versión de Delphi 4 y con ella, tal y como nos tiene acostumbrados, muchas mejoras y novedades. Interfaz mejorada, creación asistida de código, soporte de CORBA, drivers nativos de Access 97 y Oracle 8, lenguaje más potente...

I*nprise* (el nuevo nombre de *Borland*) lanzó en Junio del presente año la nueva versión de Delphi. Para los que programamos en Delphi desde que salió la primera versión, resulta curioso ver cómo va evolucionando una herramienta que fue tan completa desde el principio. En Delphi 4 el conjunto de novedades es muy amplio y el desarrollador de cualquier versión de Delphi, agradecerá las nuevas características que conlleva la actualización. Habrá que ver cuál es la respuesta del gran competidor en el mercado *RAD* (*Rapid Application Development,* Desarrollo Rápido de Aplicaciones), es decir, Visual Basic, aunque lo tendrá difícil, ya que Delphi, con igual facilidad de programación, consigue toda la potencia de un compilador de C++.

Las mejoras realizadas abarcan todos los aspectos del compilador, desde el propio lenguaje a la interfaz, pasando por la depuración y el soporte de tecnologías de programación distribuida.

En el apartado de la interfaz de desarrollo se ha ampliado el *Code Insight* (Visión del Código), con más facilidades de escritura. A las ayudas contextuales se añaden otras opciones interesantes como el acceso directo a la definición completa de los objetos.

El lenguaje *Object Pascal* también ha sido remozado, con posibilidades de sobrecarga de métodos (al más puro estilo de C++ o Java), arrays dinámicos (es decir, de tamaño inicial no prefijado) y nuevos tipos de datos.

El acceso a tipos y objetos de *Oracle 8*, además de los *drivers* nativos para esta y otras bases de datos (*Informix*, *Sybase*, *Microsoft SQL Server*), confieren un acceso muy sencillo a los datos, con independencia de la base que esté siendo utilizada.

Una de las novedades más importantes aparece en la versión Cliente/Servidor -destinada a empresas y profesionales que quieran acercarse a la flexibilidad de las aplicaciones con arquitectura distribuida o basadas en la web-. En esta versión se habilita la programación distribuida independientemente de que el programador opte por utilizar el estándar abierto *CORBA* (*Common Object Request Broker Architecture,* Arquitectura de Agentes de Petición de Objetos Comunes) o el propietario de Microsoft (COM o DCOM, *Distributed Common Object Model.,* Modelo de Objetos Comunes Distribuidos). Incluso se puede optar por dar soporte simultáneo a ambos. El *ORB* (*Object Request Broker*) de Visigenic se incluye también para el diseño y prueba de los objetos CORBA. La depuración se puede realizar ya de forma remota desde cualquier puesto con conexión *TCP/IP* con el que ejecute la aplicación.

El lenguaje Object Pascal también ha sido remozado en esta ocasión

La opción Cliente/Servidor añade componentes *HTTP*, *FTP* y *POP3* a los componentes de *sockets* de la versión 3, además de componentes de servidor

Tabla 1. Características más importantes de cada una de las versiones de Delphi 4.

CARACTERÍSTICA	Delphi 4 Estándar	Delphi 4 Profesional	Delphi 4 Cliente / Servidor
Número de componentes de la VCL (Visual Component Library)	85	100	175
- Desarrollo de aplicaciones de 16 y 32 bits	●	●	●
- Creación visual de componentes y Herencia visual de Formularios	●	●	●
- CodeInsight (Asistente para facilitar la escritura de código) y Code Templates (plantillas de código)	●	●	●
- IDE configurable a nuestro gusto, ventanas adosables	●	●	●
- Nuevo Administrador de proyectos, que permite controlar varios proyectos al tiempo	●	●	●
- Sobrecarga de métodos, arrays dinámicos, enteros de 64 bits	●	●	●
- Ayudante para completar el código y las clases con la sintaxis correcta, y para indicar los parámetros de los métodos	●	●	●
-Depuración de DLLs; empleo de colores en depuración para determinar la sintaxis; depuración multiproceso	●	●	●
- Vista de los registros internos de la CPU	●	●	●
- Librerías de paquetes; Componentes de Windows 95 y Windows 98. Almacén de formularios prediseñados, Expertos y Módulos de datos	●	●	●
- Explorador de código, que esquematiza el código de un proyecto, con enlaces a los símbolos e "historial" de navegación (como en un navegador de Internet)	○	●	●
- OpenHelp para configurar los contenidos de la ayuda	○	●	●
- Inspector de depuración (para controlar las propiedades de los componentes), Monitor de mensajes de Windows (WinSight) y "log" de eventos	○	●	●
- Código fuente de la VCL; "unit" con funciones estadísticas y financieras.	○	●	●
- Editor de colecciones de paquetes de componentes	○	●	●
- Gestor de paquetes de componentes	●	●	●
- Controles de la VCL "adosables" (como ventanas); limitación del tamaño máximo y mínimo de los componentes así como de sus posiciones relativas en pantalla.	●	●	●
- Lista de Acciones "compartidas" por varios componentes	●	●	●
- Soporte de varios monitores (Windows 98); barras de desplazamiento planas	●	●	●
- Componentes: calendario Windows 98, Página desplazable W98; Barra de controles; TeeChart (gráficos "vivos")	○	●	●
- QuickReport (para implementar informes fácilmente)	●	●	●
- Soporte de documentos activos, Interfaces COM, Controladores y servidores OLE	●	●	●
- Creación inmediata de ActiveForm y ActiveX; Ayudante de Objetos COM	○	●	●
- Soporte JPEG y de imágenes fractales	○	●	●
- Componentes "sockets"; Componentes cliente de Internet; Desarrollo web; ActiveForms; BDE en formato CAB para su distribución fácil a través de la web	○	●	●
- Drivers para: Access 97, dBase, FoxPro, Paradox	●	●	●
- Conectividad ODBC completa y API de acceso al BDE (Borland Database Engine, Motor de bases de datos de Borland)	○	●	●

Tabla 1. Características más importantes de cada una de las versiones de Delphi 4. (Continuación)

- Drivers nativos SQL: Oracle, MS SQL Server, Informix v9, Sybase, DB2 (IBM) e Interbase	○	○	●
- Componentes conectados con BDs; actualizaciones de BDs almacenadas en caché para mejorar la eficiencia	●	●	●
- Explorador y Administrador de bases de datos	●	●	●
- Explorador SQL para visualizar procedimientos almacenados y disparadores, Monitor SQL para depurar aplicaciones y Constructor de SQL			●
- Licencia de Local InterBase para un usuario	○	●	●
- Licencia de InterBase NT para 5 usuarios	○	○	●
- Soporte avanzado de Oracle 8	○	○	●
- Arquitectura de Bases de datos multinivel (Multi-Tier), que facilita el mantenimiento de los clientes.	○	○	●
- Soporte de CORBA y de Microsoft Transaction Server	○	○	●
- Servicios de Windows NT	○	○	●
- Aplicaciones de Servidor Web; WebBridge (solución abierta que da soporte a ISAPI y a NSAPI); WebModules (para publicar información); WebDispatch (para responder a los clientes); WebDeploy (para generar clientes "delgados" de configuración cero)	○	○	●
- Depuración remota de procesos, desde cualquier punto donde exista una conexión TCP/IP con la máquina donde se estén ejecutando	○	○	●
- VersionManager, para el control de las versiones;	○	○	●
- Compilador de ayudas	●	●	●
- Soporte Multi-byte (para desarrollos internacionales) y texto bidireccional	●	●	●
- InstallShield, para realizar programas de instalación con todos los ficheros necesarios;	○	●	●
- Edición de los formularios de Delphi sin el código fuente	○	●	●

● Si ○ No

con interfaces *ISAPI* y *NSAPI*, o *CGI* o *WinCGI*.

Veamos más detenidamente alguna de las nuevas características de Delphi 4.

¿QUÉ DELPHI NECESITAMOS?

Lo primero que debemos saber antes de acercarnos a la tienda a comprar Delphi 4 es qué ofrece cada una de las versiones del mismo. La primera de ellas, la *Standard* cuesta 18.595 pta. (15.995 para estudiantes) y tiene lo necesario para dominar Delphi y desarrollar cualquier tipo de aplicación monopuesto (es decir, que no sea distribuida).

La versión *Professional* está pensada para el profesional de la programación que, además de mejoras y más componentes, necesita disponer del código fuente de la Librería de Componentes, o del programa *InstallShield* (que permite realizar instalaciones con todos los ficheros necesarios para que las aplicaciones funcionen correctamente) entre otros. Esta versión cuesta 105.000 ptas.

Por último, la versión *Client/Server* está destinada a empresas o profesionales que necesiten desarrollar aplicaciones distribuidas o basadas en la web, por un precio de 399.000 ptas. Esta última cuenta con un número mayor de componentes, Objetos *CORBA* y *DCOM*, más herramientas. Además existe una última versión, denominada *Enterprise* y que saldrá a la venta en Diciembre, que une la versión Cliente/Servidor con el software de *RPC* (*Remote Procedure Call*, Llamada a Procesos Remotos) **Entera 4**.

En la Tabla 1 se pueden apreciar las diferencias entre cada una de las versiones disponibles en el mercado. Veamos a continuación los detalles de algunas de las características más importantes.

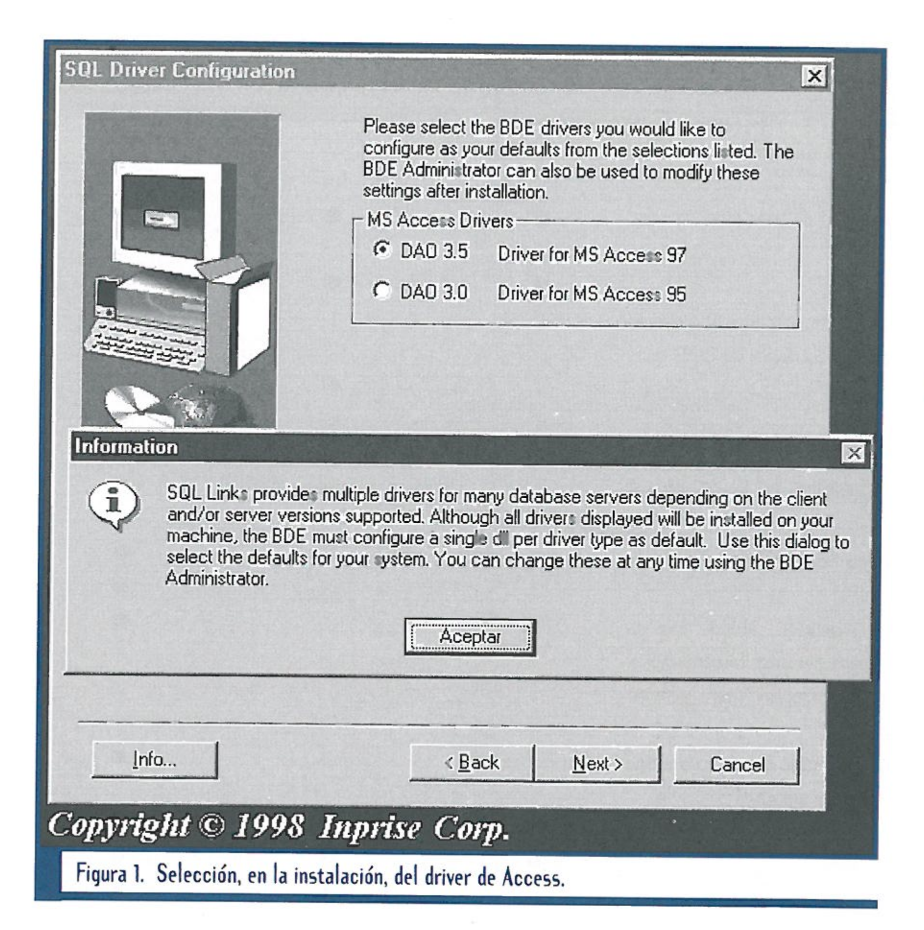


Figura 1. Selección, en la instalación, del driver de Access.

INSTALACIÓN

Respecto al proceso de instalación de Delphi, (en concreto de la versión *Standard* resulta conveniente comentar que durante la misma se nos pregunta por el *driver* de *Access* que queremos instalar: el de *Access 97* o el de *Access 95*. Aquí seleccionaremos el de la base de datos que vayamos a emplear pues sólo se crea una *DLL* para cada *driver* de base de datos (aunque se puede cambiar una vez instalado Delphi). En la figura 1 se observa la pantalla de selección de este *driver* en la instalación y la explicación de la necesidad de elegir uno u otro.

Al final de la instalación se pregunta al usuario si desea actualizar la librería **COMCTL32.DLL**, ya que según se informa será necesaria para el correcto funcionamiento de Delphi. La razón es que Delphi 4 soporta las nuevas características de Windows 98 (como la opción multi-monitor), y esa *DLL* (obtenida del Internet Explorer 4.01) contiene lo necesario para emular el funcionamiento requerido de este nuevo sistema operativo (recordemos que, simplificando un poco, Windows 98 = Windows 95 + Internet Explorer 4).

MEJORAS EN EL IDE

El IDE (*Integrated Development Environment*, Entorno Integrado de Desarrollo) guarda semejanza con el de la versión 3, aunque en realidad ha cambiado mucho. Lo primero que apreciamos es una apariencia distinta de los botones de las barras de herramientas. Si en Delphi 3 se añadieron los botones "planos", ahora se han añadido además las barras flotantes - para adaptarse a los tiempos - que permiten situar cada una en el orden y la posición que queramos. En la figura 2 se muestra la diferencia visual entre ambas.

Ya es posible configurar las barras de herramientas y ocultar las que queramos

Esta presentación da una pista sobre una nueva característica que se echaba en falta en versiones anteriores: la posibilidad de configuración de los botones de las barras de herramientas, así como de las barras que queremos mostrar u ocultar. Además es posible la creación de barras de herramientas personales. Para conseguirlo, tendremos que seleccionar las opciones **View/Toolbars/Customize**. En el diálogo que aparece a continuación (figura 3), podremos arrastrar botones hacia y desde las barras de herramientas a este cuadro de diálogo.

Otro aspecto bastante novedoso son los menús con imágenes (reflejo de la nueva propiedad que incorporan los componentes de menú de la *VCL*, pues como ya sabemos, el propio Delphi está desarrollado con Delphi). En la figura 4 podemos apreciar la mejora visual entre un menú de Delphi 4 y el mismo en Delphi 3.

El amontonamiento de ventanas en pantalla se resuelve gracias a la posibilidad de adosarlas

El amontonamiento de ventanas que usualmente se formaba en Delphi cuando abríamos varias utilidades (por ejemplo, el *Object Inspector*, el *Project Manager* y el visor de *Breakpoints*) se mejora ahora con la posibilidad de adosar ventanas a la izquierda de la venta-

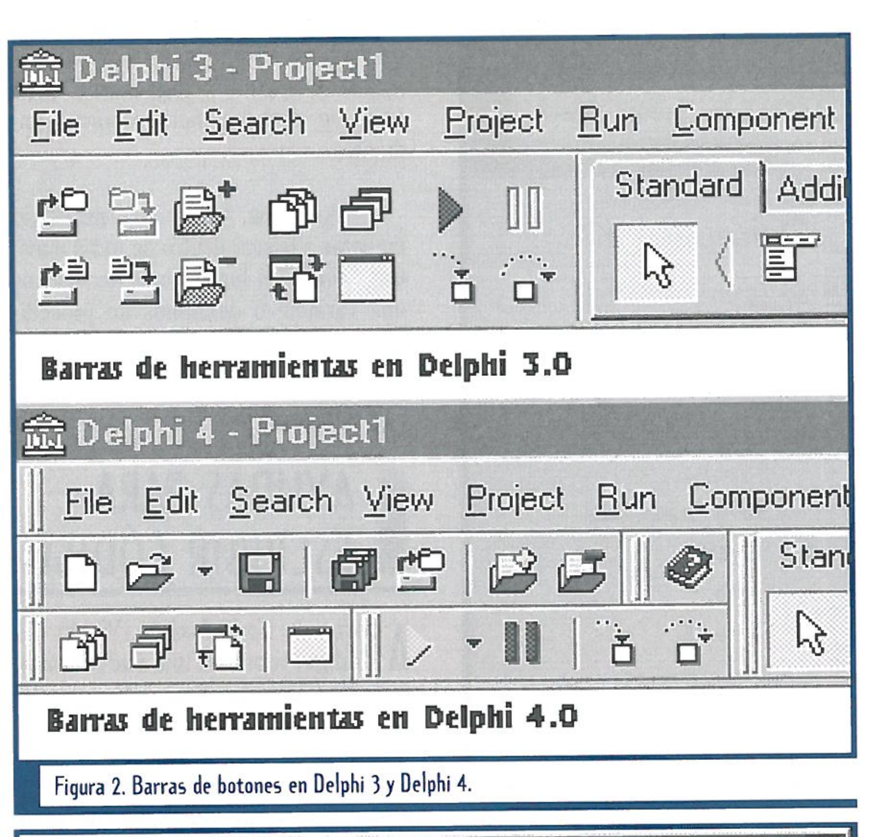


Figura 2. Barras de botones en Delphi 3 y Delphi 4.

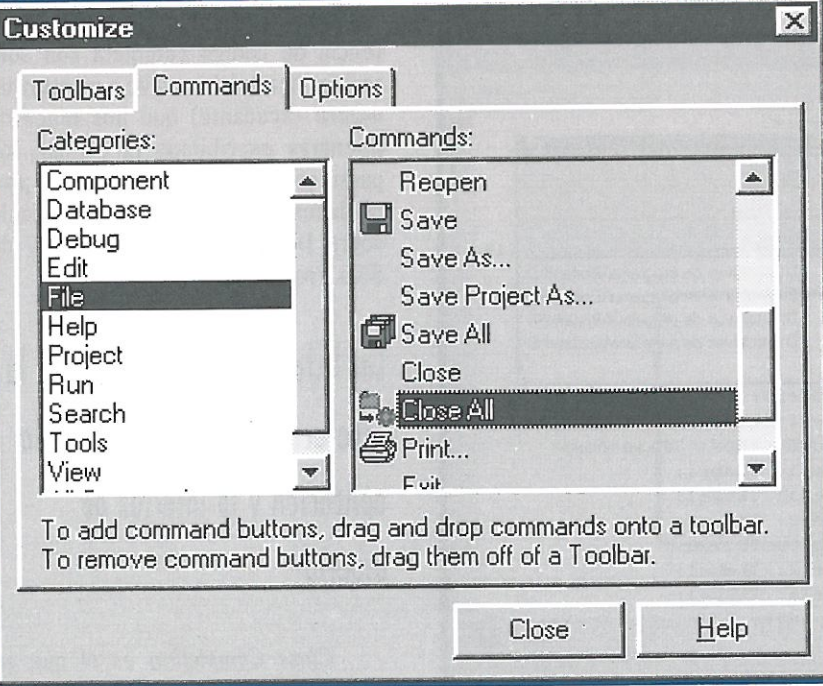


Figura 3. Cuadro de diálogo para personalizar los menús y barras de herramientas.

na de edición. Para observar este funcionamiento, sólo tenemos que desplazar el *Object Inspector* hacia el borde izquierdo de la ventana de edición, y aparecerá un cuadro de borde difuminado que nos indicará donde puede "anclarse" la ventana. Si abrimos el *Project Manager* - Administrador de proyectos - (menú **View/Project Manager**), aparecerá una nueva ventana superpuesta a las anteriores, de forma parecida a como se muestra en la figura 5.

Si desplazamos esta ventana y situamos el cursor debajo del *Object Inspector,* podremos anclar el *Project Manager* debajo del anterior (ver figura 6).

El Administrador de Proyectos puede gestionar varios proyectos al mismo tiempo

El *Project Manager* es uno de los elementos que ha sido mejorado, siendo ahora capaz de gestionar varios proyectos al tiempo. Por ejemplo, se puede gestionar un proyecto y las dos *DLLs* que se manejan desde el mismo. Además, es posible compilar todo el conjunto de una vez utilizando la secuencia de opciones **View/Build All Projects**. Otra mejora relacionada con la gestión de los proyectos es que si hacemos doble clic desde el Explorador de Windows, en un proyecto de Delphi y Delphi 4 ya está ejecutándose, no arrancará una nueva sesión del compilador, sino que se abrirá en la que ya se encuentra en ejecución.

Code Insight permite completar una sentencia de código con sólo teclear una palabra clave o mostrar un asistente

Las versiones *Professional* y *Client/Server* añaden una nueva herramienta muy útil: el Explorador de Código (*Code Explorer*). En esta herramienta se muestra un esquema con todas las clases y sus procedimientos, métodos, tipos, variables y *"uses"*, de un fichero **.pas*. En la figura 7 se puede ver el contenido que muestra esta herramienta.

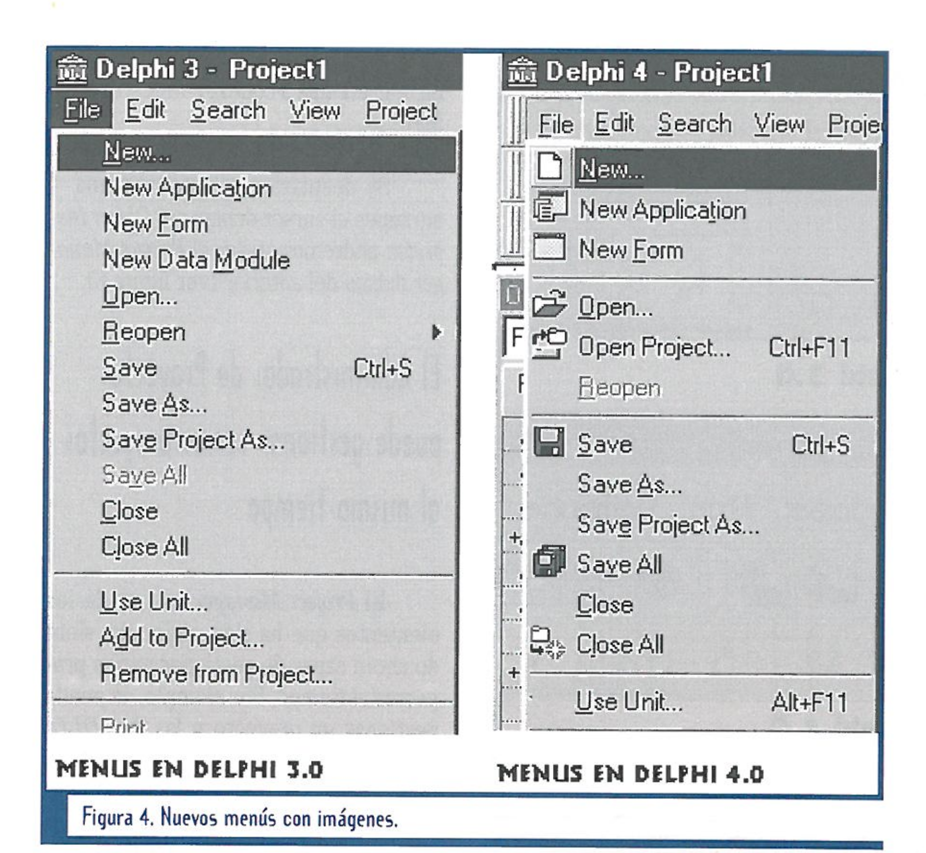


Figura 4. Nuevos menús con imágenes.

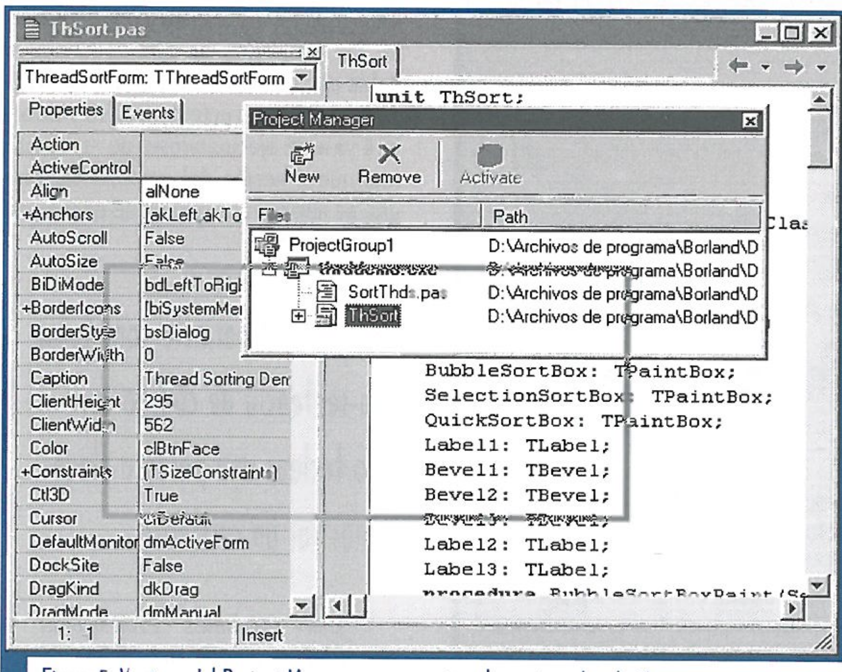


Figura 5. Ventana del Project Manager superpuesta a la ventana de edición.

Por ejemplo, haciendo doble clic en *TCustomDBGrid*, se abre la rama correspondiente y aparecen tres subramas: *Private, Protected y Public.* Haciendo doble clic en *Private,* se abrirá la subrama correspondiente. Ya sólo habrá que seleccionar el método, la propiedad o la variable deseada y en la ventana de edición (ver figura 8) se mostrará el código donde esté definida (como a partir de la versión Professional, se incluye el código fuente de la *VCL*, se abrirá el fichero correspondiente).

Así pues, ya no será necesario empezar a buscar dentro de todas nuestras *"Units"* el lugar donde declaramos una variable o definimos un procedimiento.

AYUDAS PARA ESCRIBIR CÓDIGO

Aparte de *Code Insight* (Visión del Código) tenemos una nueva herramienta: *Class Completion* (Relleno del Código de una Clase), que lamentablemente sólo se encuentra disponible en las versiones *Professional* y *Client/Server*. Recordamos que *Code Insight* permite completar una sentencia de código completa con sólo teclear una palabra clave o mostrar un *wizard* (ayudante) que nos muestra, mientras escribimos el código, los parámetros de un método o las propiedades de una clase (ver artículo sobre Delphi 3 en el número 37 de Sólo Programadores).

Las listas de acciones facilitan la separación entre la lógica de la aplicación y la interfaz de usuario

Class Completion es el que se encarga de generar el código básico de una clase a partir de un esbozo de la misma. Rellenando la declaración de una clase, esta herramienta rellena el código básico en la implementación y viceversa. La mejor manera de ver cómo funciona es hacerlo mediante un ejemplo.

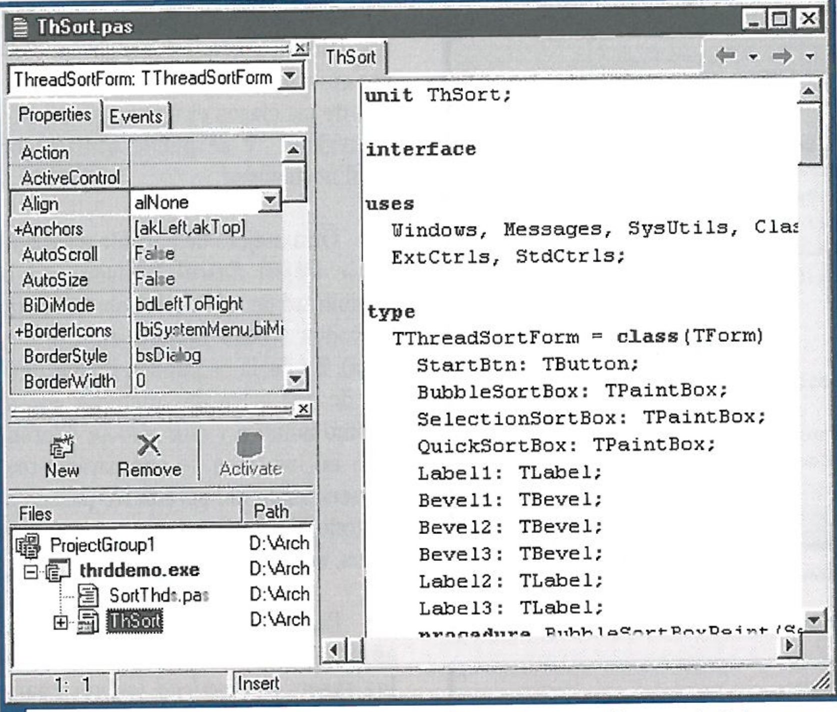


Figura 6. Ventana del Project Manager adosada al Object Inspector a la izquierda de la ventana de edición.

Introducimos las siguientes líneas de código:

```
TPrueba = class (TComponent)
  property PropiedadUno: integer;
  procedure MetodoUno (Parametro: integer);
end;

implementation

function TPrueba.MetodoDos: integer;
begin
  Result := PropiedadUno;
end;
```

La mejora más práctica reside en la introducción de los arrays dinámicos

Al pulsar el botón derecho, y seleccionar en el menú contextual **Complete Class at Cursor** se crea:

```
TPrueba = class (TComponent)
private
  FPropiedadUno: integer;
  procedure SetPropiedadUno(Value: integer);
public
  property PropiedadUno : integer read FPropie-
      dadUno write SetPropiedadUno;
  procedure MetodoUno (Parametro: integer);
  function MetodoDos: integer;
end;

implementation

function TPrueba.MetodoDos: integer;
begin
  Result := PropiedadUno;
end;

procedure TPrueba.MetodoUno (Parametro:
    integer);
begin

end;

function TPrueba.SetPropiedadUno (Value: inte-
    ger);
begin
  FPropiedadUno := Value;
end;
```

LISTAS DE ACCIONES

Dado que varios controles pueden ejecutar el mismo comando (por ejemplo un botón y una opción de menú que terminen el programa), se ha diseñado un sistema para separar los elementos de la interfaz de usuario de la lógica de la aplicación. La idea es simple: se crea un componente *TActionList* donde se almacenan todas las acciones "compartidas" (*TAction*), y desde un editor especial, asignamos las acciones a todos los controles que queramos. En la figura 9, a la izquierda aparece el método tradicional de asignación de eventos y a la derecha el funcionamiento mediante las listas de acciones.

MEJORAS DEL LENGUAJE

La mejora más práctica se traduce en la introducción de *arrays* dinámicos. Con este nuevo tipo podemos

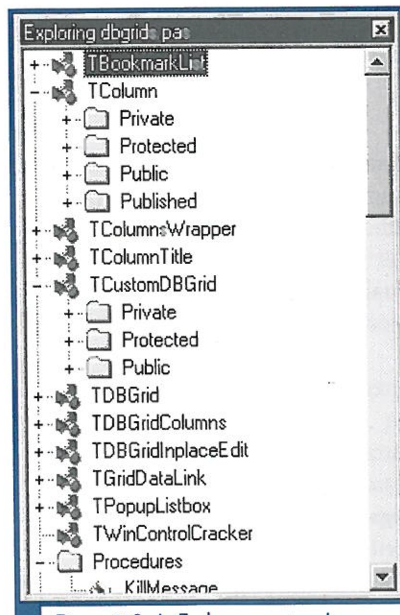


Figura 7. Code Explorer mostrando información del código de un proyecto.

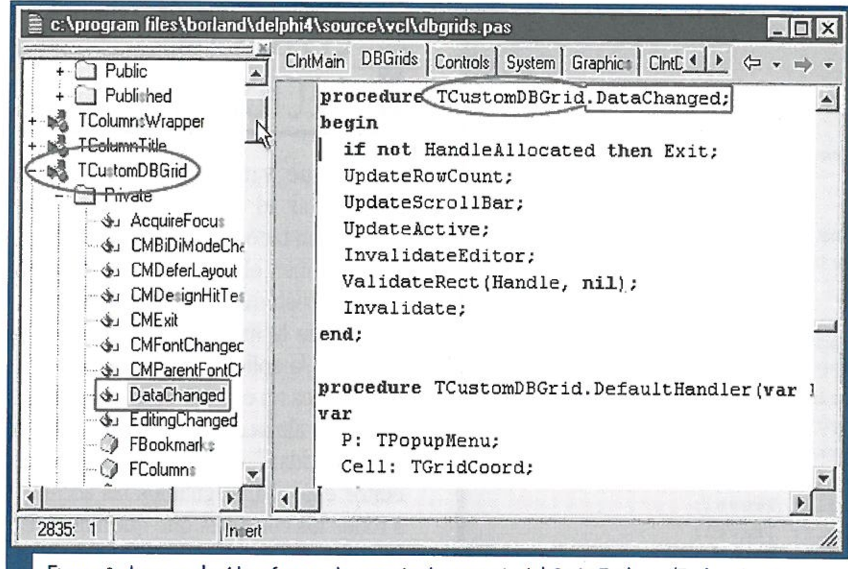


Figura 8. Acceso al código fuente de un método a través del Code Explorer (Explorador de Código).

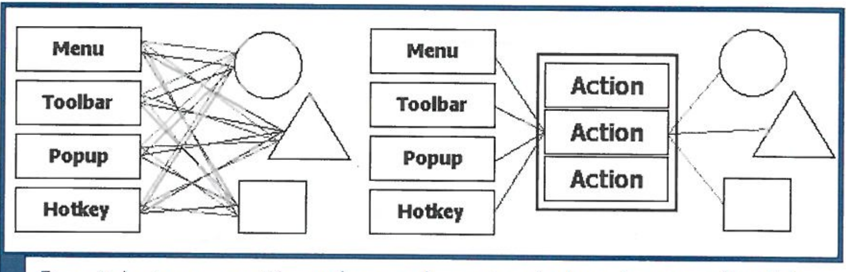


Figura 9. Acciones compartidas por los controles gracias a las listas de acciones (derecha).

olvidarnos de los conflictivos punteros para muchas de las tareas en las que los emplearíamos.

Ahora basta definir un *array* abierto (es decir, sin especificar el número de elementos) y utilizar la función *SetLength* para poder establecer y cambiar el tamaño del mismo en cualquier punto del ejecutable. Un ejemplo:

```
Procedure TForm1.PruebaArrayVariable;
var
 ArrayVariable: array of integer;
 i: integer;
begin
 SetLength (ArrayVariable, 5);
 For i:= 0 to 4 do begin
  ArrayVariable[i] := i;
  Edit1.Text := Edit1.Text + ' ' +
       IntToStr(ArrayVariable[i]);
 end;
 SetLength (ArrayVariable, 10);
 For i:= 0 to 9 do begin
  ArrayVariable[i] := i;
  Edit2.Text := Edit2.Text + ' ' +
       IntToStr(ArrayVariable[i]);
 end;
end;
```

Class Completion se encarga de generar el código básico de una clase a partir de un esbozo de la misma

Otra mejora es la introducción de enteros de 64 bits y de enteros sin signo (*cardinal*) de 32 bits.

De igual manera, el tipo *Real*, que daba problemas y obligaba a utilizar en la implementación de propiedades de las clases el tipo *single* ha sido "arreglado" y se puede utilizar con total normalidad.

Otra mejora destacable en el lenguaje *Object Pascal*, consiste en la posibilidad de utilizar la sobrecarga de métodos (como ocurre en C++ y Java). Es decir, se pueden definir dentro de una clase dos métodos con el mismo nombre y que sólo se diferencien en los tipos de sus parámetros. Dependiendo del parámetro pasado al método, se llamará a uno o a otro (o a otros, si son más de dos).

Para sobrecargar un método, se añade al final de la declaración la palabra reservada *overload* (sobrecargar). Por ejemplo:

```
TPrueba = class (TComponent)
private
 procedure Repetido (Entero: integer); overlo-
     ad;
 procedure Repetido (Cadena: string); overload;
...

{ En un procedimiento de otra clase }
var
 prueba: TPrueba;
begin
 prueba:= TPrueba.Create;
 prueba.Repetido(1);    { Repetido(Entero) }
 prueba.Repetido('dos'); { Repetido(Cadena) }
 prueba.Free;
end;
```

CONCLUSIÓN

Tan sólo hemos podido ver algunos de los aspectos más importantes y destacables de entre el conjunto de novedades de Delphi 4. Esto sólo ha supuesto un acercamiento a las muchas ventajas que os puede llegar a proporcionar en vuestro trabajo una herramienta como ésta.

Programación del API de Windows (y IV)

Jordi Agost (agost@eup.udl.es)

Con este artículo damos por finalizada la serie y acabamos con una colección de ejemplos prácticos sobre el mundo de la API de Windows una vez que ya conocemos la teoría de las entregas anteriores. Aprenderemos a conectarnos o desconectarnos de una red o a que un formulario aparezca siempre visible.

En unas pocas líneas vamos a ver un ejemplo que nos va a ser muy útil, especialmente si somos administradores de una red, o simplemente trabajamos con ella. Mostraremos cómo podemos conectarnos o bien desconectarnos de unidades de discos a través de una red. Para ello vamos a necesitar las declaraciones siguientes obtenidas a través del visor API, una vez que se conoce la función que queremos. La primera nos servirá para conectarnos a una unidad de red y la segunda para desconectarnos de dichas unidades:

```
Declare Function WNetConnectionDialog Lib
        "mpr.dll" Alias "WNetConnectionDialog"
        (ByVal hwnd As Long,
ByVal dwType As Long) As Long

Declare Function WNetDisconnectDialog Lib
        "mpr.dll" Alias "WNetDisconnectDialog"
        (ByVal hwnd As Long,
ByVal dwType As Long) As Long
```

Después, una vez que hemos añadido en la sección de declaraciones, debemos de insertar dentro del formulario creado al efecto un par de botones (supongamos que les damos el nombre **Command1** y **Command2** respectivamente), y acto seguido nos queda añadire el siguiente código para el primero de ellos, que se encarga de efectuar la conexión:

```
Private Sub Command1_Click()

Dim x As Long
  x = WNetConnectionDialog(Me.hWnd,
        RESOURCETYPE_DISK)

End Sub
```

Y el siguiente para el botón Command2, mediante el cual se realizará la desconexión:

```
Private Sub Command1_Click()

Dim x As Long
  x = WNetDisconnectDialog(Me.hwnd,
        RESOURCETYPE_DISK)
End Sub
```

UN FORMULARIO SIEMPRE VISIBLE

El siguiente ejemplo nos permitirá poner un determinado formulario siempre visible por encima de cualquier otro. De esta forma, aunque dicho formulario no tenga el foco permanecerá por encima del que lo tiene (si sus coordenadas en pantalla son las mismas). De esta manera no sólo cambiará la propiedad que permite tener un *form* por encima de los otros, sino también su tamaño y su localización. Vale la pena mirar con detenimiento la

siguiente función, debido a las posibilidades que tiene. Su declaración es la siguiente:

```
Declare Function SetWindowPos Lib "user32"
        Alias
"SetWindowPos" (ByVal hwnd As Long, ByVal
        hWndInsertAfter As Long,
ByVal x As Long, ByVal y As Long, ByVal cx As
        Long, ByVal cy As Long,
ByVal wFlags As Long) As Long
```

Esta función permite establecer un nuevo estado y una nueva posición para una ventana. También se podrá cambiar la posición de la ventana en la lista interna de ventanas, que es precisamente lo que vamos a ver a continuación en las siguientes líneas.

El primer parámetro (*hwnd*) es el *Handle* de la ventana que vamos a manipular. El segundo (*hWndInsertAfter*) consiste en un manipulador de ventana que puede tomar uno de los siguientes valores:

- **HWND_BOTTOM:** colocará la ventana al final de la lista de ventanas.
- **HWND_TOP:** colocará la ventana en lo alto del orden-Z, el orden en que se representan en pantalla las ventanas según su nivel jerárquico.
- **HWND_TOPMOST:** colocará la ventana en lo alto de la lista, por encima de cualquier ventana de nivel superior.
- **HWND_NOTOPMOST:** colocará la ventana en lo alto de la lista, después de las ventanas de nivel superior.

El tercer y cuarto parámetro (*x* e *y* respectivamente) son las nuevas coordenadas *x* e *y* de la ventana. Si *hwnd* es una ventana hija entonces, el valor de *x* y de *y* se dará en coordenadas de cliente de la ventana madre. Los parámetros *cx* y *cy* especificarán el nuevo ancho y alto de la ventana. Por último el parámetro *wFlags* será uno de los siguientes indicadores:

- **SWP_DRAWFRAME:** dibuja un marco alrededor de la ventana.
- **SWP_HIDEWINDOW:** esconde la ventana.
- **SWP_NOACTIVATE:** no activa la ventana.
- **SWP_NOMOVE:** mantiene la posición actual (ignora los valores *x* e *y*).
- **SWP_NOREDRAW:** la ventana no se redibuja automáticamente.
- **SWP_NOSIZE:** mantiene el tamaño actual (ignorará *cx* y *cy*).
- **SWP_NOZORDER:** mantiene la posición actual en la lista de ventanas (ignora *hWndInsertAfter*).
- **SWP_SHOWWINDOW:** presenta la ventana por pantalla.
- **SWP_FRAMECHANGED:** enviará un mensaje del tipo *WM_NCCALCSIZE* a la ventana (mensaje que indica que el tamaño ha cambiado), aunque el tamaño no haya cambiado.

Los procedimientos son los siguientes:

```
Public Sub StayOnTop(hWnd As Long, Size As
        Rect)    SetWindowPos hWnd,
        HWND_TOPMOST, Size.Left / 15,
      Size.Top / 15, Size.Width / 15, Size.Height /
        15,
      SWP_NOACTIVATE Or SWP_SHOWWIN-
        DOWEnd SubPublic Sub
        NotStayOnTop(hWnd As Long, Size As
        Rect)    SetWindowPos hWnd,
        HWND_NOTOPMOST, Size.Left / 15,
      Size.Top / 15, Size.Width / 15, Size.Height /
        15,
     SWP_NOACTIVATE Or SWP_SHOWWIN-
        DOWEnd Sub
```

Observemos que en el primer procedimiento el segundo manipulador de ventana (*hWndInserAfter*) tiene el valor **HWND_TOPMOST** y en el segundo procedimiento su valor "contrario" **HWND_NOTOPMOST.**

EL NOMBRE DEL USUARIO

En el siguiente ejemplo que vamos a ver a continuación vamos a obtener el nombre del usuario que actualmente está utilizando el ordenador. Para ello utilizamos la función *GetUserName* situada en la librería de enlace dinámico **advapi32.dll**. Su declaración es la siguiente:

```
Private Declare Function GetUserName Lib
        "advapi32.dll" Alias "GetUserNameA"
        (ByVal lpbuffer As String, nSize As Long)
        As Long
```

Dicha función tiene dos parámetros: el primero es un área intermedia inicializada con la longitud *nSize* donde se cargará el nombre del usuario, mientras que el segundo (*nSize*) es una variable cuyo valor inicial es la longitud de *lpBuffer*. La función devolverá aquí la cantidad de caracteres cargados en *lpBuffer*.

```
Function GetUser()
Dim sBuffer As StringDim lSize As Long
  sBuffer = Space$(255)  lSize = Len(sBuffer)
  Call GetUserName(sBuffer, lSize)  If lSize > 0
       Then   GetUser = Left$(sBuffer, lSize)
  Else    GetUser = vbNullString
  End IfEnd Function
```

Dentro de la función comprobamos si el número de bytes devueltos es superior a cero, ya que en caso afirmativo eliminaremos los bytes innecesarios de la cadena. Suponiendo que el número de bytes devueltos fuese igual a cero, entonces la función devolvería una cadena vacía.

CONVERSIÓN DE TEXTO (OEM/ANSI)

Vamos ahora a ver un pequeño ejemplo que nos permitirá convertir texto de formato MS-DOS a texto Windows y viceversa. Es independiente que estemos utilizando Windows 3.x o Windows 9x, ya que dichos sistemas operativos usan su propio juego de caracteres (recordemos que NT se basa en el estándar *UNICODE*) basado en *ANSI* (*American National Standars Institute*). Dicho juego de caracteres se conoce comúnmente con el nombre de "juego de caracteres Windows". Existen además varias páginas de códigos *ANSI* disponibles en el sistema. Podemos saber la página que actualmente está utilizando el sistema mediante la función *GetAcp*:

```
Declare Function GetACP Lib "kernel32" Alias
    "GetACP" () As Long
```

Existen una serie de funciones que permiten convertir los caracteres del sistema *OEM* al *ANSI* y viceversa. Dichas funciones normalmente las utilizaremos cuando trabajemos con nombres de archivos, ya que en sistemas de archivos tipo *FAT* (*File Allocation Table*) siempre se utiliza el juego de caracteres *OEM*.

Es importante especificar que el mapa de conversión de un juego de caracteres a otro no es siempre uno a uno, razón por la cual si convertimos una cadena de tipo *OEM* a *ANSI* y luego hacemos el mismo proceso al revés, el resultado puede que no sea idéntico al original.

Para entender la explicación anterior, tomemos por ejemplo el carácter **K**. Si utilizamos la página de código estándar de los Estados Unidos (E.E.U.U.) veremos que dicho carácter no existe con lo que si intentamos salvar un archivo que usase este código, obtendríamos un error o en su defecto un archivo en el que en ningún caso podríamos acceder desde el teclado.

La función *CharToOem* de la *API* convertiría dicho carácter en la letra **E**, aunque por las razones explicadas anteriormente la función *OemToChar* no convertirá una letra *E* en una *K*. Un ejemplo de conversión seria el siguiente:

```
texto_dos$ ="holà, cosà, casà, imprì"
texto_oem$ = "holà, cosà, casà, imprì"
Call OemToChar(texto_dos$, texto_oem$)msg-
    box(texto_oem$)
```

INFORMACIÓN DE UNA VENTANA

Vamos ahora a ver una rutina con funciones de la *API* que puede resultar de gran utilidad si las ventanas de nuestras aplicaciones pueden ser redimensionadas por el usuario. El ejemplo que presentamos a continuación mostrará en una etiqueta el nombre de la ventana y en otra etiqueta sus dimensiones.

Para que el código funcione deberemos crear un nuevo proyecto, insertar en él un control de tiempo (*Timer*), al cual le vamos a poner el nombre **Info**. Su intervalo lo estableceremos con el valor **1000**. A continuación añadiremos dos etiquetas (*Label*) que llevaran los nombres **lblNombreVentana** y **lblDimensiones** respectivamente.

Posteriormente, en el evento *Timer* del objeto añadiremos el siguiente código:

```
Private Sub tmrInfo_Timer()

Dim hd&, res&, H&, W&, Length%, Stri$
Dim Size As RECT

  hd& = GetForegroundWindow&()
  Long% = GetWindowTextLength&(hd&)
  Long% = Long% + 1
  Stri$ = String$(Long%, 0)
  res& = GetWindowText&(hd&, Stri$, Long%)
  lblNombreVentana.Caption = Stri$
  res& = GetWindowRect(hd&, Size)
  H& = Size.Bottom - Size.Top
  W& = Size.Right - Size.Left
  lblDimensiones.Caption = "Height " & H& &
      " Width " & W&

End Sub
```

Con dicho procedimiento lo que obtenemos, en primer lugar, es el manipulador de la ventana en primer plano, es decir, la ventana activa de la aplicación en primer plano. Y la función devuelve un valor de tipo *Long* que es el manipulador de la ventana en primer plano.

Posteriormente con la función *GetWindowTextLength* recuperamos la longitud del texto de título de una ventana (aunque también podríamos hacerlo con contenido de un determinado control).

Debemos hacer dos puntualizaciones sobre esta función: en primer lugar que la longitud que devuelve la función no incluye el carácter **NULL** del final de la cadena. En segundo lugar, es necesario indicar que es preferible no usarla para formularios o controles de VB, siendo mucho más recomendable utilizar las propiedad *Caption* o *Text* de los mismos.

Seguidamente, al contador de longitud le añadimos lo que ocupará el carácter **NULL**.

Inicializamos la cadena que contendrá el valor del título de la ventana y con *GetWindowText* recuperamos el título de la misma, mostrándolo en la etiqueta correspondiente.

A continuación lo que hacemos es tomar las dimensiones de la pantalla (con *GetWindowRec*) y las visualizamos, sacando la información de la estructura de tipo *RECT* que le hemos pasado a dicha función.

FUNCIONES GENERALES DE INFORMACIÓN DEL SISTEMA

Vamos a ver ahora una serie de funciones que nos van a permitir obtener información acerca del sistema Windows y controlar algunas de sus características principales. Observaremos como algunas de estas funciones se encargan de duplicar algunas de las funcionalidades del panel de control de Windows.

- **GetLastError:** Obtiene una cierta información ampliada del error producido en la última llamada a una función de la *API*.
- **GetSysColor:** Determina el color vigente para un objeto Windows .
- **GetSystemInfo:** Recupera información sobre el hardware.
- **GetSystemMetrics:** Determina ciertos parámetros del sistema, tales como la altura de la barra del menú, el ancho de la barra de desplazamiento vertical, etc.
- **GetSystemPowerStatus:** Recupera información sobre la fuente de energía y del estado del sistema.
- **GetVersion, GetVersionEx:** Determina las versiones de DOS y Windows que se están usando.
- **GetWinFlags:** Determina el modo de operación de Windows y el tipo de *CPU* y de procesador .
- **SetComputerName:** Establece el nombre del ordenador.
- **SetEnvironmentVariable:** Establece una variable de entorno dentro del bloque de entorno.
- **SetSysColors:** Establece los colores vigentes para el objeto Windows (por ejemplo el color de una barra de desplazamiento).
- **SystemParameterInfo:** Se trata de una de las funciones más potentes y sirve para recuperar y fijar un gran número de parámetros de Windows como algunos de los que citamos a continuación.

Por ejemplo simplemente modificando el primer parámetro de la función (parámetro llamado *uAction*) podemos recuperar los valores siguientes: información sobre la minimización y maximización de ventanas (*SPI_GETANIMATION*), información sobre tiempos de accesos (*SPI_GETACCESSTIMEOUT*), para controlar el tamaño del borde de una pantalla (*SPI_GETBORDER*), para saber si con el arrastre se mueve toda la pantalla o no (*SPI_GETDRAGFULLWINDOWS*), información sobre si está activado o no el intercambio rápido de tareas (*SPI_GETTASKSWITCH*), sobre el granulado de rejilla (*SPI_GETGRIDGRANULARITY*), accesos a la pantalla y a la escala y disposición de los iconos (*SPI_GETHIGHCONTRAST* y *SPI_GETICONMETRICS*), la fuente de títulos de los iconos (*SPI_GETICONTITLELOGFONT*), la demora de teclado (*SPI_GETKEYBOARDDELAY*) y su repetición (*SPI_GETKEYBOARDSPEED*), información sobre el acceso al ratón (*SPI_GETMOUSEKEYS*), el modo de apagado de pantalla (*SPI_GETPOWEROFFTIMEOUT*), la activación del salvapantallas (*SPI_GETSCREENSAVEACTIVE*), el tiempo para activarse del mismo (*SPI_GETSCREENSAVETIMEOUT*), la velocidad del doble clic (*SPI_SETDOUBLECLICKTIME*), el mapa de bits utilizado como fondo (*SPI_SETDESKWALLPAPER*), para activar el modo de ahorro de energía (*SPI_SETLOWPOWERACTIVE*), para activar el modo de ahorro de energía en pantalla, para intercambiar los botones del ratón (*SPI_SETMOUSEBUTTONSWAP*), para activar el salvapantallas (*SPI_SETSCREENSAVEACTIVE*)... y conste ante todo que se han citado aproximadamente la mitad de parámetros que contiene).

DIRECTORIO DE WINDOWS Y DEL SISTEMA

Vamos ahora a utilizar algunas de las funciones anteriormente descritas para obtener información, por ejemplo para conocer el directorio de sistema y el directorio de Windows utilizaremos el siguiente código:

```
Dim windirec&, winsystem&
windirec = GetWindowsDirectory(buffer$,
    500)Msgbox buffer$winsystem =
    GetSystemDirectory(sysbuffer$,
    500)Msgbox sysbuffer
```

MEMORIA LIBRE

De una forma análoga podemos obtener la memoria libre de la que dispone nuestro sistema, para ello como observamos, declaramos la siguiente estructura y la siguiente variable:

```
Dim memsts As MEMORYSTATUSDim
    memory As Long
```

Luego hacemos la llamada a la función de la *API*:

```
GlobalMemoryStatus memsts
```

Y a continuación mostramos la información que dicha función nos da:

```
mem = memsts.dwTotalPhysMsgBox "Total M.
    física – " & Format$(mem \ 1024,
    "###,###,###") & "K"mem& =
    memsts.dwAvailPhys
MsgBox "M. física disponible – " &
    Format$(mem \ 1024,
```

```
    "###,###,###") & "K"mem& =
    memsts.dwTotalVirtualMsgBox "Total M.
    Virtual - " & Format$(mem \ 1024,
    "###,###,###") & "K"mem& =
    memsts.dwAvailVirtual
MsgBox "M. Virtual Disponible - " & For-
    mat$(mem \ 1024, "###,###,###") &
    "K"
```

INFORMACIÓN DEL SISTEMA

También podemos extraer la información referida al sistema Windows, sobre el cual se está ejecutando nuestro programa, para ello hacemos las declaraciones pertinentes:

```
Dim verinfo As OSVERSIONINFO Dim build
    As String, ver_major As String,
    ver_minor As String Dim ret As Long
    Dim MSG As String
```

Y de forma análoga las declaraciones de la *API*. Hay que decir que de forma incomprensible el visor de *API* no contempla la función *GetVersionEx* que va a ser la que vamos a utilizar para nuestro ejemplo, de todas formas ahí va la declaración:

```
Declare Function GetVersion Lib "kernel32" ()
    As Long
Declare Function GetVersionEx Lib "kernel32"
    Alias "GetVersionExA" (lpVersionInfor-
    mation As OSVERSIONINFO) As Long
```

Posteriormente donde nos interese colocaremos el siguiente código:

```
verinfo.dwOSVersionInfoSize = Len(verinfo)
 ret = GetVersionEx(verinfo)
 If ret = 0 Then
  MsgBox "Error al obtener la versión de Win-
    dows "
  End
 End If
 Select Case verinfo.dwPlatformId
  Case 0
   MSG = MSG & "Windows 32s "
  Case 1
   MSG = MSG & "Windows 95 "
  Case 2
   MSG = MSG & "Windows NT "
 End Select
 ver_mayor = verinfo.dwMajorVersion
 ver_menor = verinfo.dwMinorVersion
```

Si estamos utilizando Windows 98, lo va a detectar como si fuese Windows 95, con la diferencia de que el número de versión será menor (la variable *ver_menor*) va a tomar el valor **10**. Es decir, Microsoft al pasar de Windows 3.x a Windows 95, consideró que pasaba de la versión 3.x del producto a la 4.0. Y al pasar de Windows 95 a 98 considera que pasa de la versión 4.0 a la 4.x.

PROCESADOR DEL SISTEMA

También podemos deducir el tipo de procesador de la máquina utilizada. Para ellos añadiremos las siguientes constantes en la sección de declaraciones de un módulo:

```
Public Const PROCESADOR_INTEL_386 =
    386
Public Const PROCESADOR_INTEL_486 =
    486
Public Const PROCESADOR_INTEL_PEN-
    TIUM = 586
Public Const PROCESADOR_MIPS_R4000 =
    4000
Public Const PROCESADOR_ALPHA_21064 =
    2106
```

Y la declaración (por razones de espacio obviamos la de la estructura *SYSTEM_INFO*):

```
Declare Sub GetSystemInfo Lib "kernel32"
    (lpSystemInfo As SYSTEM_INFO)
```

Posteriormente en el sitio en que nos interese añadimos el siguiente código:

```
Dim sysinfo As SYSTEM_INFO

 GetSystemInfo sysinfo
 Select Case sysinfo.dwPROCESADORType
  Case PROCESADOR_INTEL_386
   MsgBox "Intel 386"
  Case PROCESADOR_INTEL_486
   MsgBox "Intel 486"
  Case PROCESADOR_INTEL_PENTIUM
   MsgBox "Intel Pentium"
  Case PROCESADOR_MIPS_R4000
   MsgBox "MIPS R4000"
  Case PROCESADOR_ALPHA_21064
   MsgBox "DEC Alpha 21064"
  Case Else
   MsgBox "Procesador Desconocido"
 End Select
```

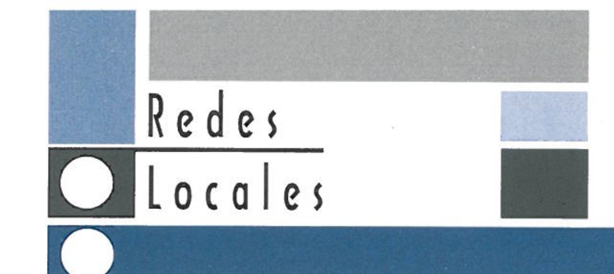
Debemos decir que si se realiza la prueba con un equipo con procesador *Pentium II*, la rutina lo detecta como si fuese un *Pentium* (según las declaraciones, el valor **586**).

ción simplemente seleccionamos el tipo que hemos elegido:

```
Select Case tipodisco  Case
    DISCO_TIPO_INDETERMINADO
  MsgBox "Unrecognized"  Case DISCO_
    RAIZ_INEXISTENTE    MsgBox
    "doesn't exist"  Case DISCO_CDROM
    MsgBox "CD-ROM"  Case DISCO_FIJO
    MsgBox "I.E. Hard Disk"  Case
    DISCO_RAMDISK   MsgBox "RAM
    disk"  Case DISCO_REMOTO   Msg-
    Box "I.E Network drive."  Case
    DISCO_REMOVIBLE   MsgBox "I.E.
    Floppy Disk."  End Select
```

CONCLUSIÓN

Cerramos de este modo esta serie sobre el mundo de la *API* de Windows, abordado desde el punto de vista del Visual Basic. En sus dos primeros capítulos aprendimos los fundamentos teóricos sobre los accesos de la *API*, y posteriormente en este capítulo y en el anterior hemos desarrollado varios ejemplos prácticos para entender mejor su funcionamiento.



Interconexión de redes (I)

Enrique Díaz Trobo (enriqued@jet.es)

A lo largo de este artículo vamos a ver de qué modo se comportan los diferentes tipos de protocolo, tales como los de contienda, polling, y token. Asimismo, realizaremos una primera aproximación a los dispositivos que nos sirven para poder interconectar redes.

En el artículo sobre *Cableado de Red* del número 49 vimos, entre otras cosas, algunos aspectos relacionados con el funcionamiento de los protocolos. En estas páginas avanzaremos algo más sobre este particular, intentando abordar con más detalle los dispositivos necesarios para la conexión de equipos.

CONTROL DE LA COMUNICACIÓN

El proceso de transmisión de datos implica una serie de procedimientos que abarcan desde el nivel físico hasta la presentación de la información en un formato determinado. En esta ocasión vamos a ver más detenidamente el nivel de enlace, o lo que es lo mismo el control de la comunicación a través de los protocolos. Ya vimos como toda comunicación se divide en tres fases: establecimiento de la comunicación, transferencia de la información y finalización de dicha comunicación. La manera de establecer y terminar la comunicación depende de cómo estén conectadas las dos estaciones de trabajo, es decir, si están conectadas a través de un cable de serie o paralelo, una línea punto a punto, etc.

La forma en que se va a controlar el tránsito de la información depende única y exclusivamente del protocolo empleado. Este protocolo que hemos elegido controlará la comunicación y, por tanto, deberá llevar a cabo las siguientes funciones:

- Sincronización la comunicación.
- Control de los errores de transmisión.
- Coordinación de la comunicación.
- Detección y recuperación de los posibles fallos que se se lleguen a producir a lo largo del proceso de comunicación.

Cuando transmitimos información, ésta se distribuye en bloques de una longitud determinada que están dispuestos en un orden determinado. Además, existirá un control de errores que ha de permitir la comprobación de que todos y cada uno de los bits enviados sean iguales a todos y cada uno de los bits recibidos. Al realizar la comunicación de este modo conseguimos que si se produce un error en alguno de los bloques de información enviados, sólo será necesario volver a transmitir el bloque defectuoso, evitando repetir toda la transmisión hasta que la información se reciba correctamente.

PROTOCOLOS

Los protocolos que nos permiten efectuar esta tarea son los que describimos a continuación:

- Contienda
  - Contienda simple

- Acceso múltiple por detección de portadora **(CSMA)**
- Acceso múltiple por detección de portadora con detección de colisiones **(CSMA/CD)**
- Acceso múltiple por detección de portadora evitando colisiones **(CSMA/CA)**

- *Polling* (llamada selectiva)
- *Token Passing* (Paso de testigo)

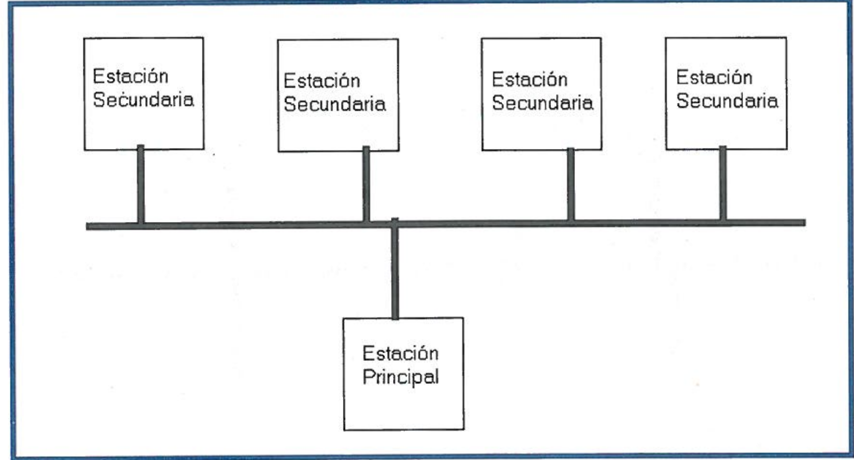


Figura 1. En los protocolos Polling las estaciones sólo podrán transmitir si la estación principal lo permite.

LOS PROTOCOLOS DE CONTIENDA

Contrariamente a lo que se suele creer, una y sólo una estación puede transmitir a través de un cable a un mismo tiempo. Así pues, hay que establecer unas normas que permitan regular el acceso de cada estación a la red.

La forma en que se controla el tránsito de la información depende exclusivamente del protocolo utilizado

Un protocolo de contienda es aquél cuyo método de acceso al cable consiste en que el primero que llega es el primero que lo utiliza. Por eso mismo se llama "de contienda", ya que en una red muy activa hay verdaderos "tortazos" por conseguir la línea. Tanto es así que una red que maneja protocolos de contienda puede llegar a colapsarse con las "discusiones" sobre cómo se organiza el tráfico en la red.

CONTIENDA SIMPLE

En este protocolo todas las estaciones comparten el mismo canal de transmisión y los mensajes se envían a través de dicho canal; es decir, tenemos un solo cable al que acceden muchas estaciones. Dicho de otra manera, se trata de un solo cable con acceso múltiple. Las estaciones responden únicamente a aquellos mensajes que incluyen su dirección, mientras que el resto son ignorados. Mientras no reciban un mensaje que incluya su dirección, se encuentran en estado de espera pero escuchando el canal de transmisión.

Por lo tanto, podemos tener dos situaciones: que las estaciones se encuentren transmitiendo datos o bien que se encuentren en estado de espera. Las estaciones envían los bloques de datos sin controlar antes si el canal está disponible o no. De este modo, cuando un bloque de una estación coincide con el de otra se produce una colisión y ambos bloques se destruyen automáticamente. Si el bloque llegara a su destino, la estación receptora enviaría un mensaje indicando que lo ha recibido. Si la estación emisora, después de un lapso de tiempo determinado, no ha recibido el mensaje indicando la correcta recepción, volverá a repetir la transmisión del bloque que haya finalizado la transmisión de datos.

Por las razones antes indicadas, este tipo de protocolo no es recomendable en redes con cargas medias o altas de tráfico, ya que se estarían produciendo colisiones constantemente y el rendimiento de la red sería muy bajo y tendría tiempos de espera muy grandes. Debemos tener en cuenta que con este tipo de protocolo se produce el efecto de "bola de nieve": cuanto más tráfico hay, más colisiones se producen, lo que provoca que se incremente todavía más el tráfico debido a las repeticiones, lo que a su vez genera un mayor número de colisiones...

ACCESO MÚLTIPLE POR DETECCION DE PORTADORA

En este protocolo también se emplea un único canal al que acceden muchas estaciones, pero las estaciones no transmiten hasta que el canal está libre. Para ello, la estación escucha el tráfico de la red para saber si hay otra estación que esté enviando un bloque de datos. Existen dos modos de realizar la escucha:

- Escuchar continuamente una frecuencia secundaria especial a la espera de que quede libre y entonces transmitir. Esto es lo que se llama detección continua de portadora.
- Se produce la escucha, y si el canal está ocupado, se abandona el intento de transmisión duran-

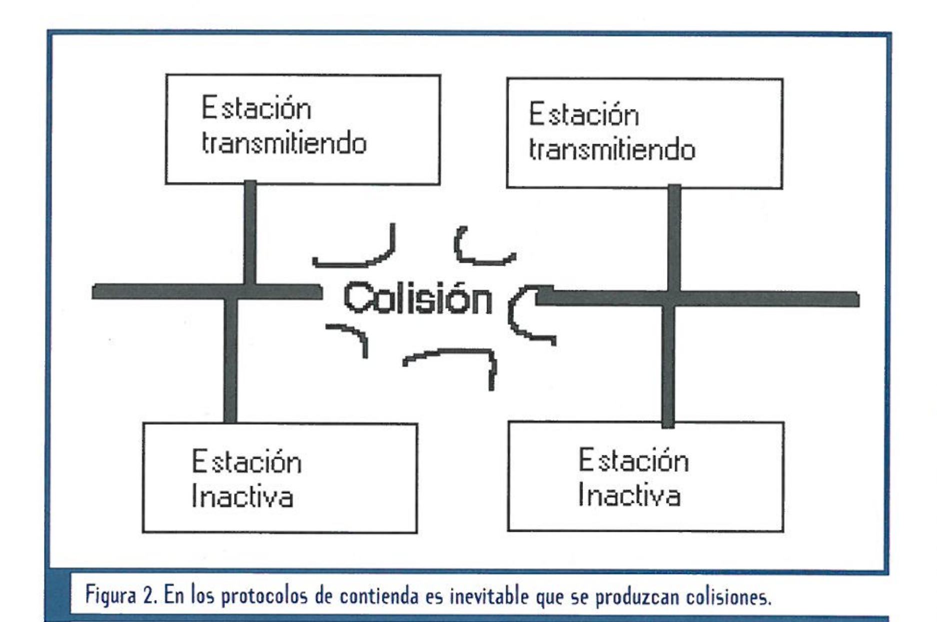


Figura 2. En los protocolos de contienda es inevitable que se produzcan colisiones.

te un lapso de tiempo aleatorio. Después, se produce una nueva escucha y comienza de nuevo el proceso. Esto es lo que se denomina como detección no continua de portadora.

Una red puede llegar a colapsarse sólo con el tráfico generado por las colisiones, es decir, a causa de la contienda

Si la escucha, en cualquiera de sus modos, revela que la línea está libre se envía el bloque de datos y, además, otra señal en la frecuencia secundaria para notificar a las demás estaciones que la línea está ocupada. Una vez que se ha transmitido el bloque de datos, la estación emisora espera hasta recibir el mensaje de que la estación receptora ha recibido el bloque.

Si este mensaje no se recibe o se recibe una señal negativa, la estación supone que se ha producido una colisión, es decir, que dos estaciones emisoras han detectado al mismo tiempo que la línea está libre y han realizado un envío simultáneamente. En este caso, se esperará un lapso de tiempo aleatorio y volverá a intentar enviar el bloque de datos.

Así pues, tenemos tres posibles estados de las estaciones: que se estén transmitiendo datos, que se encuentren en estado de espera o bien que se estén escuchando la línea.

Este protocolo es adecuado para redes de tráfico medio o bajo. Además, los segmentos de red deberían tener una longitud no muy grande, ya que de este modo el tiempo que tarda la señal en propagarse es pequeño, y el riesgo de que dos estaciones decidan enviar bloques de datos simultáneamente y colisionen será menor.

ACCESO MÚLTIPLE POR DETECCIÓN DE PORTADORA CON DETECCIÓN DE COLISIONES (CSMA/CD)

Este protocolo actúa de la misma manera que el anterior pero, además de comprobar si la línea está libre antes de comenzar la transmisión, comprueba si se ha producido alguna colisión durante la transmisión. En el caso de que así sea detiene la transmisión y se vuelve a enviar el bloque de datos después de un tiempo de espera aleatorio. Este protocolo obtiene un rendimiento mayor que el de los dos anteriores, lo que le hace adecuado para longitudes medias de segmento de red y tráfico medio o bajo.

ACCESO MÚLTIPLE POR DETECCIÓN DE PORTADORA EVITANDO COLISIONES (CSMA/CA)

Este tipo de protocolo actúa de forma análoga al anterior, con la diferencia de que cuando una estación va a enviar un bloque de datos comprueba que la línea está libre y, si es así, indica a las demás estaciones que va a efectuar una transmisión. Cuando hay varias que están esperando, la transmisión se realiza por turno.

En los protocolos Token las estaciones no pueden transmitir si antes no han capturado el testigo

En el turno de transmisión se tiene en cuenta tanto la prioridad de la estación como el orden en el que se ha indicado que desea transmitir. Así pues, a igual prioridad, transmitirá primero la que lo haya solicitado antes. Si una estación tiene una prioridad más baja transmitirá después aunque haya solicitado antes la transmisión que otra de mayor nivel de prioridad. El rendimiento de este tipo de protocolo es mayor que el de los tres anteriores, por ello es adecuado para cargas de tipo medio o alto y para una longitud media de la red.

LLAMADA SELECTIVA (POLLING)

Para poder utilizar este protocolo deberemos definir una estación

como "estación principal" mientras que el resto serán "estaciones secundarias". Las estaciones secundarias disponen de una zona de almacenamiento temporal, donde se guardará el bloque de datos que se pretende transmitir.

La estación principal comprobará cada una de las estaciones secundarias para ver si alguna tiene algún bloque de datos que transmitir. Cuando lo encuentra en alguna de ellas, se le autoriza a dicha estación para que lo transmita de forma inmediata, o bien transcurrido un determinado tiempo.

Los bloques de datos se pueden enviar de dos maneras: pasando por la estación principal, la cual los reenviará a la estación destino o bien enviándolos directamente a la estación de destino.

Cuando una estación quiere transmitir, primero ha de esperar a que llegue hasta ella el testigo vacío

Se puede establecer el mismo nivel de prioridad para las estaciones secundarias, o bien hacer que las estaciones que cuentan con una mayor actividad tengan una prioridad más alta. Así se paga con un mayor rendimiento a las estaciones más "trabajadoras". También podemos establecer que las estaciones que no estén activas no se controlen por parte de la estación principal.

Este tipo de protocolo aporta las siguientes ventajas con respecto a los de contienda:

- Se pueden enviar más datos en cada bloque.
- Soporta mayores cargas de tráfico en la red.
- Los segmentos de red pueden ser mayores.

Estas características hacen este protocolo adecuado para redes con carga media y para una longitud media o grande de los segmentos de la red.

PASO DE TESTIGO (TOKEN PASSING)

Este protocolo hace circular continuamente por la red un único testigo en forma de grupo de bits. Este testigo está formado por una cabecera, un campo de datos y un campo final (figura1).

Cuando una estación quiere transmitir, primero ha de esperar a que llegue hasta ella el testigo vacío. Cuando lo captura, lo modifica añadiéndole unos valores, de tal forma que el testigo queda formado por la cabecera, la dirección de destino, la dirección de origen, el camino que ha de seguir para llegar a su destino y el bloque de datos, y lo envía a su destino (figura 1). Si la estación no pretende transmitir, pasará el testigo vacío a la siguiente estación y así sucesivamente.

El testigo lleno llegará a la estación destino que extraerá el bloque de datos y pondrá una marca en el testigo indicando si lo acepta o bien lo rechaza por tener errores. Una vez marcado, devuelve el testigo a la estación que lo ha enviado.

Cuando el testigo llega a la estación que lo envió, ésta puede tomar una de estas tres acciones: lo devuelve si llega con la marca de rechazado, envía el siguiente bloque de datos o vacía el testigo para que pase a la estación siguiente.

Este protocolo aporta las siguientes ventajas:

- Elimina por completo el riesgo de colisiones, ya que existe un solo testigo y ninguna estación puede transmitir sin tener el testigo vacío.
- Es posible enviar gran cantidad de datos en cada bloque.
- Es apropiado para redes con grandes volúmenes de carga.
- El tamaño de los segmentos de la red puede ser grande.

Este protocolo es adecuado para redes con volumen de carga medio o alto y para una longitud media o grande de la red.

CONTROL DE ERRORES

Otra fuente de problemas en la transmisión es debida a interferencias que aparecen en la línea, con la consiguiente corrupción de datos al llegar a la estación destino. Los datos pueden haber sufrido alguna modificación y no coincidir exactamente con los que fueron emitidos.

Para detectar estos errores se emplean diversos métodos, que dependerán del protocolo elegido.

En los protocolos Polling se establecerá una estación principal que hará las veces de `guardia de tráfico'

Normalmente, la estación destino no corrige los bloques de datos erróneos, sino que se limita a detectar la existencia del error e informar a la estación emisora, enviando una petición para que vuelva a emitir dicho bloque de

datos. Los métodos más comunes para el control de los errores son el método de paridad y el método de redundancia cíclica.

MÉTODO DE PARIDAD

Este método consiste en añadir un bit a cada carácter enviado (el famoso bit de paridad que debemos establecer o no en las comunicaciones por módem). El método de paridad funciona de la siguiente manera: el valor que tomará este bit será tal que haga que el carácter enviado, contando el bit de paridad, tenga un número par de bits con valor uno (si es que la paridad debe ser par) o que tenga un número impar de unos (en el caso de la paridad impar, aunque suene raro).

La estación destino realizará por su parte su propio cálculo de paridad, recalculando el bit de paridad. En el caso de que el valor calculado coincida con el correspondiente a la paridad utilizada, se considera que la transmisión ha sido correcta; pero si no ha sido así, solicita a la estación emisora que repita el envío.

Este tipo de paridad (par o impar) también recibe los nombres de bit de paridad transversal, bit de paridad vertical o comprobación de redundancia vertical (VRC).

MÉTODO DE REDUNDANCIA CÍCLICA

Este método consiste en que la estación emisora agregue al final de cada bloque de datos una información calculada de acuerdo con una formula polinómica, cuyas variables son los ceros y unos enviados en el bloque de datos. Como en el caso anterior, la estación destino realiza el mismo cálculo. Si se produce el mismo resultado la transmisión es correcta, pero si no ha sido así, solicita a la estación emisora que repita el proceso.

La información añadida se llama Código de Redundancia Cíclica (*CRC*), Carácter de Comprobación de Bloque (*BIC*) o simplemente Redundancia.

INTERCONEXIÓN DE REDES

Una vez que hemos visto más o menos cómo funcionan los protocolos, comenzaremos a estudiar cómo funcionan una serie de "cacharros" (concentradores, puentes, etc.) que resultan necesarios para interconectar y segmentar las redes.

En el artículo sobre *Cableado de Redes* del número 49 vimos el concepto de segmentación, pero lo recordaremos diciendo que se trata de una técnica que permite dividir una red en subredes para mejorar el rendimiento de la red global. Los motivos por los que se segmenta una red se producen fundamentalmente por dos motivos: la ampliación de una red y el aumento del tráfico en la misma, ya sea por el tipo de información (por utilizar multimedia haciendo uso de videos, sonidos, etc.), o sencillamente porque nuestra red ha crecido y se ha producido un aumento considerable de usuarios.

Una red local consiste en un único segmento de red con su propia dirección de red que está sujeta a ciertas limitaciones. Por ejemplo, un segmento con cable coaxial fino tiene una limitación de 185 metros si utilizamos topología *Ethernet*, o 305 metros si empleamos *Arcnet*, si bien es cierto que podremos aumentar estas distancias utilizando repetidores. Si la empresa donde se va a implantar la red tiene un número elevado de puestos se superarán estos límites, con lo cual necesitaremos aumentar la distancia y capacidad de nuestra red. Para poder hacerlo, necesitaremos *bridges* (puentes), *routers* y otros dispositivos de conmutación para expandir nuestra red o interconectarla con redes de otros departamentos.

BRIDGES, ROUTERS, CONMUTADORES...

Los repetidores, puentes, *routers*, conmutadores y, en cierto modo, los concentradores son los dispositivos que podemos utilizar para ampliar la red o interconectarla con otras redes. La clave consiste en saber qué dispositivo resulta más adecuado para cada caso y situación.

Un repetidor se limita a amplificar las señales, y no cambia en modo alguno los propios paquetes ni realiza ningún filtrado

En redes pequeñas, lo más normal es que sólo deseemos incrementar el tamaño y limitaciones de distancia de nuestra red. Para solucionar este caso tan sólo debemos comprar un repetidor y conectarlo a otro segmento de cable. Un repetidor se limita a amplificar las señales de la red, de modo que podemos utilizar segmentos de cable más largos. Hacen eso y

suscríbete

a Sólo Programadores

y consigue un magnífico descuento

suscripción normal

ahorro **20%**

12 revistas (1 año) por sólo...

9.350 ptas.

suscripción estudiantes

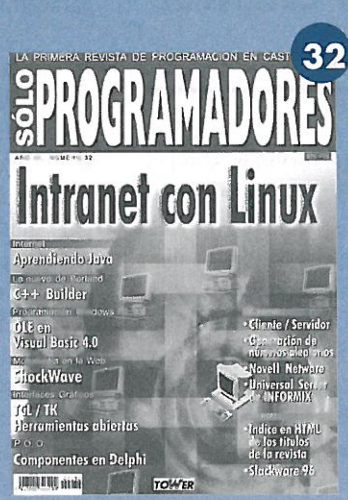
(carreras técnicas)

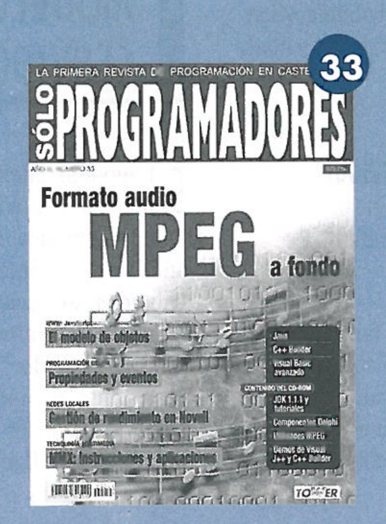
ahorro **40%**

12 revistas (1 año) por sólo...

7.050 ptas.









Programación Distribuida Orientada a Objetos con CORBA (I)

Antonio Ruiz Cortés (aruiz@lsi.us.es) y José Luis Alvarez Macías (alvarez@uhu.es)

La programación distribuida orientada a objetos basada en la tecnología CORBA se perfila como uno de los paradigmas que seguir en el desarrollo de nuevas, o adaptación de antiguas, aplicaciones corporativas que tienen como principal requisito, el de su utilización en Internet o Intranet.

Este artículo, es la primera parte de una serie, que intentará presentar los aspectos más interesantes de la programación distribuida orientada a objetos basada en la tecnología CORBA.

El modelo cliente/servidor supuso un importante avance

En esta primera entrega tan solo se da una idea general de donde está enmarcada esta tecnología, sus antecedentes, algunos aspectos sobre su estado actual y beneficios que esperan obtenerse.

EVOLUCIÓN DEL MODELO CLIENTE/SERVIDOR

El modelo cliente/servidor supuso un significativo avance en la Industria Informática de principio de los años 80. Actualmente, los objetos distribuidos son considerados como **"la nueva revolución de la revolución cliente/servidor"** [Orfali 96]. En este caso, tanto el cliente como el servidor son desarrollados basándose en componentes inteligentes con capacidad de interoperar a través de redes de comunicación. Entre las principales causas que han propiciado este nuevo paradigma podemos destacar tres:

- Abaratamiento del ancho de banda de las redes de área extensa.
- Desarrollo de sistemas operativos de sobremesa con características que, hasta hace poco, estaban reservadas a los mainframes tales como: conexión a red, multiproceso, multiusuario, etc.
- Aparición de requisitos en el desarrollo de aplicaciones cliente/servidor.

Algunos aspectos que resultan claves para satisfacer estos nuevos requisitos o necesidades son:

- **Mecanismo avanzado de transacciones**
  Las transacciones ya no estarán limitadas a un único servidor, por lo que será necesario disponer de un modelo de transacciones que pueda extenderse a varios servidores, soporte la larga vida de las mismas cuando se trasladen entre éstos, permita su almacenamiento y su utilización de una manera segura.

- **Agentes móviles**
  En el inminente mercado electrónico, jugarán un papel clave los que se han dado en llamar agentes móviles. De este modo, por ejemplo, será posible que los clientes o consumidores dispongan de agentes personales que cuiden de sus intereses; agentes comerciales que vendan sus mercancías; agentes infiltrados o espías que busquen información que permita la elaboración de estadísticas, etc. Además, estos agentes podrán 'vivir' o existir en cualquier máquina de la red.

- **Gestión de documentos compuestos**
  El contenido de los documentos tendrá naturaleza multimedia y deberá permitirse su edición, visualización, almacenamiento, recuperación y envío desde o hacia cualquier parte de la red. Por lo tanto, la mayoría de los nodos deberán utilizar una serie de tecnologías de composición de documentos: *OpenDoc*, *OLE*, etc. En este sentido, los servidores centrales deberán tener mecanismos que permitan el almacenamiento y distribución masiva de estos documentos.

- **Componentes autosuficientes**
  Con la llegada de nuevos sistemas operativos de sobremesa dotados de funcionalidad hasta ahora reservada a los sistemas operativos corporativos, se da paso a una situación hasta ahora inusual: que la máquina de un cliente, la estación de trabajo, pueda realizar funciones hasta ahora reservada a los servidores. Una de las desventajas, es la necesidad de gestión y configuración de los componentes para cada sistema operativo: DOS, Windows 3.x, 95, NT, Macintosh, Linux, OS/2, Solaris, Next, etc. Se puede dar una solución a este problema si se exige a los componentes capacidad de gestión y configuración autónomas.

- **Middleware inteligente**
  Se debe conseguir la ilusión de mostrar todo el sistema distribuido (puede abarcar desde una a cientos de máquinas) como un único sistema. Esta tarea la lleva a cabo lo que se conoce como *Middleware*, que básicamente permitirá utilizar el mismo sistema de nombrado para cualquier recurso de la red y el diálogo o interoperabilidad entre cualquier recurso de ésta, sin necesidad de preocuparse por el sistema de comunicaciones (pila de protocolos, medios de comunicación, etc.).

Las principales tecnologías que actualmente se utilizan en el desarrollo cliente/servidor y que intentan dar soluciones a estas nuevas necesidades son: bases de datos *SQL*, *TP Monitors* y objetos distribuidos.

BASES DE DATOS SQL

Se trata de la tecnología más utilizada en el desarrollo de aplicaciones cliente/servidor. Con objeto de conseguir un mejor rendimiento, las sentencias *SQL* suelen estar asociadas a un procedimiento que reside en el mismo servidor que la base de datos. Este procedimiento almacenado es invocado por los clientes mediante una llamada que sigue el modelo *RPC* (*Remote Procedure Call*). También se han añadido extensiones, *triggers* y *rules*, relacionadas con la integridad de datos y para facilitar la lógica a utilizar por los clientes.

El gran problema de estos añadidos al *SQL* estándar radica en la poca compatibilidad existente entre las distintas implementaciones que de los mismos realizan los fabricantes. Algunos autores indican que son cinco años el periodo que suele transcurrir entre la aparición de algún aspecto tecnológico relacionado con las bases de datos cliente/servidor y su reflejo en el estándar.

Aunque este problema del retraso en la estandarización e incompatibilidad entre distintos fabricantes de servidores de bases de datos se mitiga con la utilización del esquema de bases de datos federadas, son varias las razones que aconsejan la no-utilización de esta tecnología como el *middleware* para las aplicaciones distribuidas.

TRANSACTION PROCESSING MONITORS O TP MONITORS

Actúan como controladores de las transacciones entre los clientes y el servidor. Esta acción de control posibilita el acceso masivo de usuarios (de 500 en adelante) a aplicaciones cliente/servidor que de otra manera sería imposible en algunos casos y muy cara en otras. La idea clave no es otra que la compartición de recursos del servidor.

La no-proliferación de productos basados en esta tecnología se ha debido principalmente a dos causas: un marketing muy deficiente y una inadecuada adaptación al modelo de aplicaciones que se ejecutaban con la arquitectura cliente/servidor, por un exceso de capacidades ('venía un poco grande para su época').

Muchos fabricantes de *TP Monitors* pertenecen a la *OMG* y han participado muy activamente en la elaboración de los estándares del *CORBA Object Transaction Service*, impregnando a dicho servicio de la tecnología de los *TPM*.

CLIENTE/ SERVIDOR CON OBJETOS DISTRIBUIDOS

Tanto los servidores de base de datos *SQL* como los monitores transaccionales, sólo trasladan el desarrollo de aplicaciones monolíticas a bimonolíticas, es decir, de disponer de una macroaplicación en el *mainframe* que es accesible desde terminales a tener dos aplicaciones (una en el servidor y otra en el cliente) que vuelven a ser monolíticas. Por lo tanto, estas soluciones no terminan de solventar los problemas relacionados con este tipo de aplicaciones.

Aún cuando estas aplicaciones utilizan la tecnología de objetos, éstos no disponen de las ventajas de los componentes u objetos distribuidos. El principal inconveniente que podemos apreciar reside en la poca visibilidad de los mismos: ya que 'viven' dentro de un programa, sólo accesibles con el lenguaje que han sido compilados, y por supuesto ningún objeto externo a este programa puede saber de su existencia ni muchos menos del modelo de acceso que posee.

Sin embargo, cuando se desarrolla una aplicación o se rediseña una antigua haciendo uso de la tecnología de objetos distribuidos, se obtienen componentes autosuficientes que interactúan entre sí, ignorando las fronteras que pueden existir entre ellos: sistemas operativos, lenguajes de programación, redes de comunicación, etc. Con esta nueva filosofía de construcción de aplicaciones el concepto cliente/servidor se difumina, pues la misma máquina puede contener componentes cliente o servidor.

Es decir, con la tecnología de objetos distribuidos, la granularidad o nivel de detalle de los componentes mejora notablemente. Este aspecto facilita su distribución, pues los datos y la lógica del negocio son encapsuladas en los objetos, que además pueden ser localizados en cualquier lugar dentro de un sistema distribuido.

No obstante, para que el desarrollo de aplicaciones a medida basado en tecnología de componentes se adopte a escala industrial, es necesaria la existencia de colecciones (*suites*) de componentes. Los objetos pertenecientes a una misma *suite* están diseñados para interactuar del modo más eficiente posible, pues conocen cómo deben solicitar servicios a los demás objetos de la colección para realizar una determinada tarea.

SITUACIÓN ACTUAL

Analizar la actual situación de este sector de la industria informática resultaría muy provechoso, pero estaría fuera del ámbito del presente artículo. Sin embargo, se ha visto interesante reflexionar, sobre un aspecto cada vez más importante en la programación distribuida en general y que ocupa un lugar central en la norma *CORBA*: la **interoperabilidad.**

Resulta evidente, que las perspectivas de utilización de cualquier sistema informático, sea software o hardware, han cambiado mucho en los últimos años. De una situación en la que predominaban los sistemas propietarios con pocas posibilidades (y en el caso que existir muy caras) de interconexión y mucho menos de integración con otros sistemas, se ha pasado a una situación en la que los aspectos más valiosos y que más destacan en el marketing de cualquier producto, son la compatibilidad con estándares y la interoperabilidad con otros sistemas. En concreto, los aspectos relacionados con la interoperabilidad que son tratados por la norma *CORBA*, pueden agruparse en: software, hardware y medios de comunicación.

La necesidad de una interoperabilidad en el ámbito del Software, a su vez, puede verse desde tres distintos puntos de vista: lenguaje de implementación, sistema operativo del *host* y fabricantes o desarrolladores.

Pese al espectacular avance experimentado por los lenguajes de programación, siguen siendo utilizados y soportados, sistemas desarrollados en lenguajes calificados como obsoletos, a saber: Fortran, Cobol, Pascal, incluso C, por nombrar solo los más conocidos. En muchos casos, el funcionamiento de estos sistemas resulta correcto y el riesgo que entraña el desarrollo de un nuevo sistema equivalente al anterior con lenguajes de programación que aporten nueva funcionalidad, justifica en muy pocos casos la inversión necesaria para su migración.

Esta situación es una más de las muchas que hacen inevitable la convivencia entre distintos lenguajes de programación. Más adelante se verá el mecanismo utilizado por *CORBA* para resolver esta interoperabilidad entre

lenguajes en algunos casos muy dispares, que además es considerada como la más *universal* y *transparente* de las actualmente existentes.

Respecto a los sistemas operativos y fabricantes de componentes o librerías de clases se pueden usar argumentos similares a los utilizados en el caso de los lenguajes de programación, que llevan a conclusiones similares.

En el caso del hardware, está aumentando el nivel de heterogeneidad de las plataformas utilizadas en un determinado sistema, no siendo extraño encontrar sistemas donde al escenario habitual de *mainframe*, estación de trabajo y ordenador personal, comienzan a intervenir dispositivos clasificados como de electrónica de consumo, tales como los televisores con conexión a Internet. De hecho, la previsión de crecimiento del sector de dispositivos de electrónica de consumo con conexión a Internet refleja un mayor crecimiento que en los restantes.

La situación en el campo de los medios de comunicación es muy similar, el actual abanico de las tecnologías: Red Telefónica Básica, Red Digital de Servicios Integrados (*RDSI*), el Modo de Transferencia Asíncrono (*ATM*) y un largo etc, justifican la también necesaria interoperabilidad entre sistemas que utilizan una arquitectura de comunicaciones en principio incompatible.

¿QUÉ ES CORBA?

A estas alturas de lo que llevamos explicado, ya es posible que logremos entender en gran parte, la definición que de *CORBA nos* hace la *OMG* (*Object Management Group*) [OMG-OMA]: La **norma** CORBA especifica la **interoperabilidad** entre objetos en un entorno **distribuido heterogéneo** de manera **transparente**.

De esta definición se obtienen algunas claves para comprender mejor la norma *CORBA*:

- Al tratarse de una **norma** (al igual que el protocolo TCP/IP) pueden existir diversas implementaciones que cumplan los requisitos de compatibilidad con dicha norma, pero que difieran en cuanto a eficiencia, robustez, precio, etc.
- Define mecanismos que hacen posible la **interoperabilidad** en entornos **heterogéneos,** considerando a éstos en su triple vertiente: lenguaje, plataforma y sistema operativo, anteriormente comentada.
- La **transparencia** consigue que para el desarrollador de aplicaciones distribuidas sobre plataformas heterogéneas (u homogéneas) todos los aspectos de la red de comunicaciones, los lenguajes de implementación de los objetos remotos, los sistemas operativos de los *hosts*, etc., queden bajo la responsabilidad de la implementación de *CORBA* utilizada.
- Por último, es importante destacar que al tratarse de una norma elaborada por una organización plural, en principio independiente, los resultados que se obtienen son siempre fruto del consenso y además accesible en su totalidad.

ORIGEN Y EVOLUCIÓN DE CORBA

El *OMG* (http://www.omg.org), se fundó en 1.989, con el ánimo de fomentar la utilización de la tecnología orientada a objetos mediante el desarrollo de estándares que permitieran la reusabilidad, portabilidad e interoperabilidad del software en entornos heterogéneos y distribuidos [OMG-CORBAS], pues hasta esa fecha, las tecnologías relacionadas con la orientación a objetos no habían obtenido los beneficios que de ella se esperaban. En octubre de 1.991, el *OMG* presentó la primera versión de su norma *CORBA* (*Common Object Request Broker Architecture*).

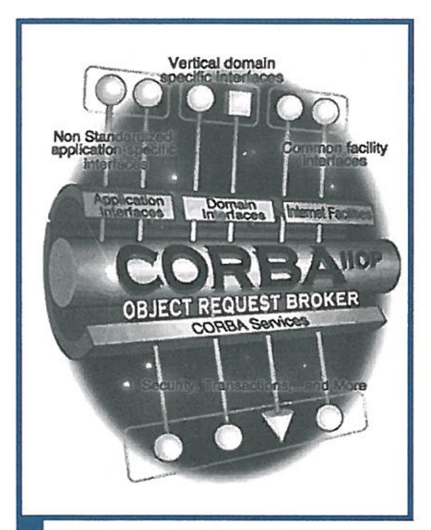


Figura 1. Logotipo de CORBA.

Todas las especificaciones elaboradas por el *OMG* tienen en común una infraestructura conceptual conocida como *OMA* (*Object Management Architecture*) sobre la que se sustentan. *OMA* intenta definir en un alto nivel de abstracción los elementos que son necesarios para hacer posible la programación distribuida orientada a objetos. *OMA* [OMG-OMA] establece una norma con cuatro categorías de interfaces: *Object Request Broker* (*ORB*), *CORBAservices*, *CORBAfacilities*, y *CORBAdomains*.

El núcleo de dicha norma es el *ORB*, un mecanismo que provee transparencia respecto a la localización, activación y comunicación con los objetos de una aplicación. Está compuesto por cinco elementos fundamentales: el núcleo del *ORB*, el Lenguaje para la Definición de Interfaces (*IDL*), las Interfaces para Invocación Dinámica

(*DII*), los Almacenes de Interfaces y de Implementaciones y los Adaptadores de Objetos [Pope 97].

La versión 1.0 de *CORBA* incluía toda la teoría del modelo de objetos *OMA*, el lenguaje *IDL* y el núcleo principal de interfaces de programación. Además, se contemplaba la gestión e invocación de peticiones dinámicas, el almacén de interfaces y el mapeo para el lenguaje C.

La mayor ventaja de CORBA sobre DCOM es el hecho de trabajar en entornos heterogéneos

La revisión 1.1 de *CORBA* apareció en febrero de 1.992, siendo la primera versión ampliamente aceptada. Corrige ciertas ambigüedades en las especificaciones originales y en el modelo de objetos, añade nuevas interfaces para el Adaptador de Objetos Básico (*BOA*) y para la gestión de la memoria y clarifica ciertos conceptos sobre el almacén de interfaces. La revisión 1.2 aparece en diciembre de 1.993, para eliminar ciertas ambigüedades en la gestión de la memoria.

ESTADO ACTUAL DE CORBA

A partir del año 1.996, la tecnología *CORBA* tomó un gran impulso, tanto en la definición de nuevas versiones de la norma como en la aparición de *ORBs*, ya sean de libre distribución o comerciales. Durante el mes de julio de dicho año, apareció la versión 2.0 en el mercado, que añadía varias características fundamentales:

- Portabilidad de los clientes
- Extensiones en el interfaz dinámico y en el almacén de interfaces
- La posibilidad de interoperar entre distintos *ORBs* a través de Internet (*IIOP*) o compatible con el *DCE* (*DCE CIOP*)
- Soporte para servicios de transacciones seguras
- Extensión de tipos para COBOL
- El mapeo con *OLE2/COM*

En Agosto de 1.997 apareció *CORBA* 2.1, que añade el *IIOP* seguro, extensiones para el lenguaje *IDL* y mapeos para COBOL y Ada. La última versión de *CORBA* hasta la fecha es la 2.2, aparecida en febrero de 1.998. Incluye la portabilidad de servidores a través del Adaptador de Objetos Portátil (*POA*) y los mapeos específicos para *DCOM* y para Java.

Aunque la arquitectura *OMA* no ha variado en lo esencial, cada uno de sus módulos ha ido experimentando una cierta evolución. La compatibilidad con la norma es fundamental para que un determinado *ORB* permita aprovechar al máximo las características de *CORBA*. Es imprescindible estudiar a fondo las posibilidades del *ORB* que vaya a usarse para una determinada aplicación y cuáles con los servicios que ofrece.

Las especificaciones de los servicios comunes que son necesarios sistemáticamente para los objetos de la aplicación también conocidas como *CORBAServices* [OMG-1995], cubren actualmente quince servicios: ciclo de vida, resolución de nombres, gestión de eventos, control de la concurrencia, serialización, relación, transacciones, persistencia, fecha, seguridad, licencia, propiedades, consultas, comercio de objetos, y gestión de configuración.

Las utilidades comunes (*CORBAfacilities* o más formalmente *Common Facilities*) son más complejas y de un nivel de detalle mayor que las *CORBAServices*. De hecho, están conformadas por servicios que son utilizados de manera conjunta con un objetivo común. También son consideradas como *application frameworks*, es decir, patrones que seguir en el desarrollo de algunos módulos comunes a un gran número de aplicaciones.

Actualmente estas facilidades están divididas en dos categorías, horizontal y vertical. Las facilidades horizontales son independientes de la aplicación. Las facilidades verticales, sin embargo, son más especificas, es decir, están orientadas a determinados conjuntos de aplicaciones.

Hasta el momento, las facilidades horizontales se han dividido en cuatro grupos: interfaz de usuario, gestión de la información, gestión de sistemas y gestión de tareas. En el caso de frecuente aparición de determinadas facilidades verticales, éstas se convertirán en horizontales, pudiéndose crear en ese caso otro grupo de facilidades horizontales.

Las facilidades verticales por su parte, han sido divididas en distintas *Domain Task Forces* (*DTF*). Cada *DTF* representa a los objetos del dominio que son específicos a un mercado vertical o dominio de problema. Las áreas que actualmente están siendo estudiadas por distintas *DTF's* son:

- Objetos de negocio
- Comercio electrónico
- Finanzas
- Manufacturación
- Medicina
- Telecomunicaciones

Los objetos de aplicaciones no están actualmente normalizados por el OMG. Son componentes de la aplicación que realizan tareas particulares para el usuario, benefician la robustez y la reutilización de los objetos del sistema. Según el *OMG Business Object Management Special Interest Group* (*BOMSIG*), un *business object* (*BO*) u

objeto de negocio, se trata de un componente del nivel de aplicación que puede ser utilizado en combinaciones impredecibles.

Por definición, un *BO* es independiente de cualquier aplicación. Las futuras aplicaciones serán conformadas por conjuntos de objetos de negocio, de manera que el desarrollo de aplicaciones se limitará al del entorno necesario para ejecutar dichos componentes.

En definitiva, una aplicación construida según las directrices de *CORBA*, consta de una gran cantidad de objetos específicos a dicha aplicación, específicos al dominio y de servicios de objetos y facilidades comunes ofrecidos por el *ORB*.

Respecto a la convivencia con Java, la versión *JDK 1.2* soporta compatibilidad con *CORBA*, además de desarrollar su propio modelo de componentes (*JavaBeans*) y la posibilidad de realizar invocaciones a métodos remotos (*RMI*).

La unión *Java-CORBA* pasa por ser una de las más interesantes dentro de la informática distribuida, ya que añade la transportabilidad del código Java y la heterogeneidad de la norma *CORBA*.

Por otro lado, el modelo *COM* distribuido (*DCOM*) es actualmente el mayor competidor de *CORBA*, aunque no son incompatibles. La mayor ventaja de *CORBA* sobre *DCOM* es la posibilidad de trabajar en entornos heterogéneos (*DCOM* es casi exclusivo de Windows), además *CORBA* permite integrar componentes *DCOM* con otro tipo de objetos (desarrollados en varios lenguajes de programación y ejecutándose sobre una gran variedad de plataformas).

Aunque resulte paradójico, *DCOM* desarrollado por Microsoft, no esta disponible para Windows 3.x, sin embargo son varios los *ORBs* disponibles para esta plataforma.

LOS BENEFICIOS DE LOS COMPONENTES DISTRIBUIDOS

No cabe duda de la creciente demanda de desarrollo de aplicaciones que hacen uso de la tecnología Internet, tanto en el desarrollo de sistemas de información corporativos privados (intranet) como públicos Internet. Existen tecnologías muy probadas y estandarizadas para la construcción de la parte estática de estos sistemas (Protocolos: *HTTP*, *SHTTP*, etc.; Lenguajes: *HTML*, *XHTML*, etc.) no siendo igual en el caso de los aspectos dinámicos y de interoperabilidad.

Los pequeños desarrolladores y vendedores de soluciones Informáticas, tendrán, gracias a los componentes, más posibilidades de introducirse en el competitivo mercado del software. Las nuevas aplicaciones distribuidas basadas en *CORBA*, asegurarán que si se desarrollan componentes compatibles la integración de éstos en aplicaciones ya existentes será una realidad. Esto permitirá una mayor independencia de los fabricantes y a la par, una mayor reutilización de lo existente.

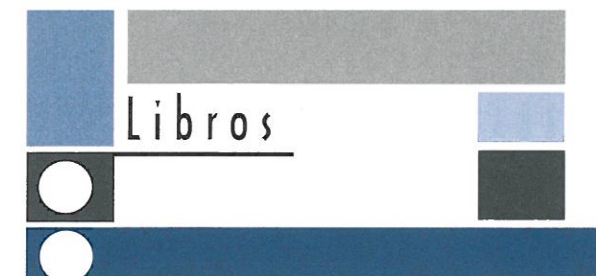
Para las grandes corporaciones, incluidas las administraciones, el disponer de una tecnología que permita el desarrollo de aplicaciones sobre Internet con menor coste económico y temporal, mayor calidad y adaptadas a cada situación, se convierte en un elemento indispensable para afrontar los retos que la siempre cambiable "Sociedad de la Información" les está imponiendo en los últimos tiempos. En este sentido, la sociedad es la primera quee va a beneficiarse de todas los teleservicios que podrán ser soportados en un futuro muy cercano, tales como: telemedicina, teletrabajo, tributación, etc.

¿POR QUÉ NO LLEGÓ ANTES?

Finalmente, puede resultar útil reflexionar sobre el estado actual de la industria del software en un aspecto en el que, todos coinciden en señalar, está en el mismo punto que estaba la industria electrónica hace 25 años. En esas fechas, se construyó el primer circuito integrado *IC*. Más adelante, se consiguió enlazar varios *ICs* sobre una misma tarjeta o placa base con el objetivo de conseguir mayor funcionalidad. Estas tarjetas se miniaturizaban en nuevos *ICs* que a su vez, volvían a ser enlazados junto a otros, sobre nuevas placas base en un proceso en espiral que aún no ha llegado a su fin.

Algunos autores consideran a los componentes como los *ICs software*, los *Framework* como placas base sobre las que se conectan éstos, y finalmente ven el bus software (*ORB*) como el bus o *backplane*. Continuando el símil, deberá ser posible la compra de *ICs* software por catálogo. Según el *Gartner Group*, esta nueva tecnología dará origen a la creación de tres nuevos mercados: desarrollo de componentes, herramientas de integración de componentes y desarrollo de aplicaciones personalizadas usando componentes.

La razón del retraso de aplicar este paradigma tan lógico, no ha sido otro que la falta de estándares, para el diseño, desarrollo y reutilización de los objetos. Son diversas las razones que han llevado a esta situación, pero se puede afirmar que hoy se cumplen los requisitos para iniciar, o mejor, continuar esta nueva revolución en el software necesaria para abordar los requerimientos de las aplicaciones actuales y futuras: un bus o canal de comunicaciones software y una tecnología de componentes que posibilita la conexión a este bus sin necesidad de adaptaciones, tipo *plug and play*.



GUÍA DE DESARROLLO DE JAVABEANS

JavaBeans es el modelo de componentes de plataforma neutral que tiene hablando a la comunidad de Java por completo. Los Beans son objetos portables reutilizables que se pueden utilizar para crear aplicaciones Java en gran escala. Este API de JavaBeans estandariza las formas en que pueden comunicarse entre sí los objetos Java, abriendo la puerta para aplicaciones en gran escala y mucho más sofisticadas.

Este libro les descubrirá a los lectores los conceptos de la programación orientada al objeto, mostrando cómo se ajustan al nuevo estándar JavaBeans, les describirá cada clase relacionada con Beans, métodos y propiedades e incluye ejemplos y proyectos sugeridos.

Además de todo lo anterior se tratan los estándar y API relacionados, proporcionando a los lectores todas las herramientas necesarias para empezar la creación de sus aplicaciones y Beans.

Conozca la migración de las APIs de gestión de red a IIOP y CORBA, descubra como funciona el nuevo modelo de eventos, aprenda y valore los pros y contras de los editores de propiedades personalizados, utilice la sección de referencia comprensiva del API de Bean y más.

Si está interesado en conocer el mundo de los JavaBeans, éste es su libro.

Editorial: Anaya Multimedia	Autor: P. Mohseni y Tom Stewart
Nº páginas: 280	Idioma: Español
Nivel: Intermedio	Precio 2.995 Ptas.(I.V.A. inc.)

CÓMO PROGRAMAR CON DELPHI PARA TORPES



Este libro, perteneciente a la conocida colección "Informática para Torpes", es otro de los manuales escritos para ayudar a dar los primeros pasos con un ordenador personal, de la manera más cómoda, fácil y divertida para aquellos usuarios que aún se resisten a sacar el mayor partido posible a su PC.

Con este ejemplar el lector obtendrá de una forma amena y original, todos los conocimientos necesarios para crear sus propias aplicaciones con Delphi sin ningún tipo de problema. Para desdramatizar el contenido, eliminando así cualquier clase de animadversión hacia la informática, se han añadido toques del mejor humor que le harán entretenida la programación.

Además de aprender, podrá divertirse al mismo tiempo, ya que los ejemplares de esta colección están ilustrados por uno de los mejores humoristas gráficos de nuestro país: Antonio Fraguas Forges.

Editorial: Anaya Multimedia	Autor: Francisco Charte Ojeda
Nº páginas: 304	Idioma: Español
Nivel: Principiante-intermedio	Precio: 2.495 Ptas.(I.V.A. inc.)



LA GUÍA LAN TIMES DE SQL

Esta es la guía completa de SQL, el lenguaje estructurado de consultas. Describe el lenguaje de base de datos especificado por el estándar ANSI/ISO SQL y utilizando actualmente por los principales sistemas de gestión de bases de datos, incluyendo SQL Server, Oracle, Sybase, DB2, Ingres y otros muchos.

Este manual le ayudará a entender los conceptos de SQL y su papel en la gestión de bases de datos. La extensa utilización de figuras e ilustraciones hace que sea más fácil de entender.

El usuario podrá utilizar SQL para recuperar y almacenar datos de ordenadores personales, redes de área local, minicomputadoras y nainframes. Será capaz de escribir programas que accedan a bases de datos SQL. Todo ello gracias a este manual.

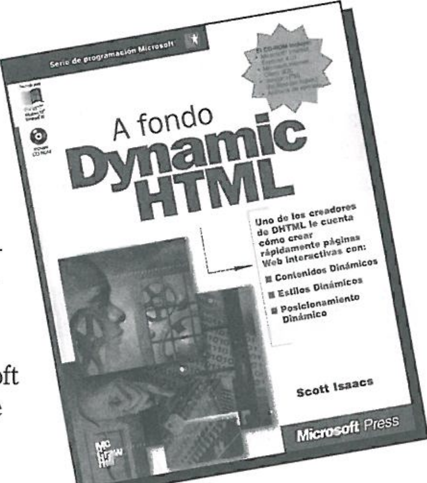
Tendrá la capacidad de seleccionar bases de datos de estas características que se adapten perfectamente a su aplicación. Las comparaciones de productos le mostrarán los puntos fuertes y las ventajas de aquellos que sean más importantes.

Editorial: McGraw-Hill
Nº páginas: 630
Nivel: Intermedio-avanzado

Autor: James R. Groff y Paul N.Weinberg
Idioma: Español
Precio: 5.000 Ptas.(I.V.A. inc.)

A FONDO DYNAMIC HTML

Las páginas web desarrolladas para Microsoft Internet Explorer 4 pueden ofrecer las características interactivas más avanzadas y excitantes gracias a Dynamic HTML, una tecnología que el autor Scott Isaacs ayudó a crear.



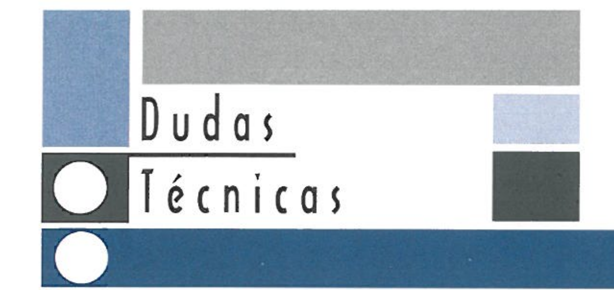
Ahora ha escrito la biblia del programador sobre este importante asunto. En parte manifiesto técnico, y en parte libro original de aplicaciones, A fondo Dynamic HTML es un manual que comienza sentando los conceptos y las herramientas básicas: HTML, hojas de estilo en cascada y los fundamentos de los guiones. Los capítulos siguientes se encargan de explicar el modelo de objetos y las colecciones de elementos.

Este libro va acompañado por un CD-ROM que proporciona una copia de Microsoft Internet Explorer 4.01, Internet Client Software Development Kit (SDK), ejemplos de guiones, etc.

A fondo Dynamic HTML es un ejemplar de gran actualidad, pensado para todos aquellos autores de páginas web, proveedores de contenido avanzados, usuarios de herramientas de guiones como VBScript y JavaScript.

Editorial: McGraw-Hill
Nº páginas: 430
Nivel: Intermedio-avanzado

Autor: Scott Isaacs
Idioma: Español
Precio: 6.000 Ptas.(I.V.A. inc.)



Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación en cualquier entorno.

Pregunta

Soy un asiduo lector de vuestra revista y he estado siguiendo vuestros artículos sobre los servlets publicados en los últimos números, y a la hora de hacer unas pruebas me he encontrado con un pequeño problema, el compilador dice que no puede encontrar los paquetes "java.servlet.Http", y "java.servlet". Creo que es un problema debido a la variable CLASSPATH.

Yo tengo instalados el JDK 1.1.5, el Java Web Server 1.1 , y también el JDK 1.2 beta4, os agradecería cualquier ayuda sobre cómo conseguir compilar mis fuentes.

Respuesta

Parece ser un problema con las variables de entorno. Comprueba que la variable de entorno *CLASSPATH* incluye los directorios de las clases del *JWS* y que la variable *JAVAHOME* contiene el directorio "raíz" correcto.

El directorio *javawebserver1.1\lib* contiene las clases *javac.jar* y *jws.jar* que son las clases requeridas, por lo que puedes ejecutar :

```
prompt>java –classpath
    C:\javawebserver1.1\lib\javac.jar; C:\javawebserver1.1\lib\jws.jar; C:\javawebserver1.1\lib\; clase
```

O modificar la variable de entorno *CLASSPATH* añadiendo el directorio C:\javawebserver1.1\lib y los ficheros *jar*. Por tanto, deberías ejecutar:

```
prompt>SET CLASSPATH=%CLASSPATH%;
    C:\javawebserver1.1\lib\javac.jar; C:\javawebserver1.1\lib\jws.jar;
C:\javawebserver1.1\lib;
```

Recuerda que se requiere tener un JDK o JRE 1.1 o superior previamente instalado para poder utilizar *Servlets* y el *JWS*. Aunque el *JWS* utiliza su propio entorno de ejecución mediante un JRE, éste ignora las variables de entorno del usuario, aunque se puede ejecutar:

```
prompt>httpd.nojre
```

Con lo que se utilizaran las variables de entorno del usuario, además de su JDK o JRE previamente instalado.

Mencionar por último, que las clases generadas en caso de utilizar *JWS* con su propio entorno tendrán que incluirse en el directorio *C:\javawebserver1.1\classes* creado por el usuario. Si se utiliza *httpd.nojre* las clases serán buscadas según el CLASSPATH del usuario.

Pregunta

Hola amigos de "Sólo Programadores", hace poco adquirí una tarjeta de vídeo modelo Amazing-1 AGP 3D con 4Mb. y con procesador Intel 740 AGP, y no consigo configurar adecuadamente el X-Windows del REDHAT 5.0. Os agradecería mucho que me dieseis alguna solución para este problema.

Respuesta

Resolver este problema te será muy sencillo. Para configurar el X-Windows, un método fácil y rápido es utilizar un programa gráfico llamado XF86Setup que Xfree86 ofrece.

Para ejecutar dicho programa únicamente deberás teclear XF86Setup con sus respectivas mayúsculas y minúsculas. En unos instantes aparecerá una pantalla con 6 menús, Mouse, Keyboard, Card, Monitor Modeselection y Other. En cada uno de ellos, deberás ajustar las opciones al hardware que dispongas.

La sección Card será la que más te interesa, ésta te permitirá poder utilizar las X-Windows si todo está bien configurado. Para ello deberás acceder a Card y de entre la lista de tarjetas gráficas soportadas, y seleccionar la que corresponda a tu modelo. Si uno no está muy seguro de la tarjeta gráfica que utiliza, se puede ejecutar SuperProbe y esperar unos instantes hasta que el programa nos imprima toda la información sobre ella. De esta forma no habrá ningún error en el momento de configurar las X-Windows.

Una vez seleccionadas todas las opciones, se deberá presionar sobre Done y esperar a ver la prueba que automáticamente realiza para asegurarse que la configuración es correcta. Si todo ha sido configurado con éxito las X-Windows informarán que ya pueden ser utilizadas. De lo contrario se deberá configurar todo de nuevo.

Pregunta

Hola, os escribo porque no se dónde acudir para resolver un problema. Estoy empezando a utilizar el sistema operativo Linux y me gustaría instalar y recompilar la versión del kernel 2.0.35 que tengo en un CD. Yo tengo la 2.0.32 instalada, pero sin las fuentes, por eso me gustaría que me indicaseis cómo debo hacerlo, partiendo de la versión que tengo en el CD, que supongo que estará comprimida, y tampoco sé ni cómo ni dónde descomprimirla. ¿Es necesario tener las versiones anteriores hasta la 2.0.35 para compilar el kernel?

Además, tengo un problema añadido, mi CD-ROM es un PIONEER DR-A24X, el cual no hay forma de montarlo, no sé si es que la versión del kernel no lo soporta o que me hacen falta unos drivers especiales, o si el CD-ROM está defectuoso. He oído algo acerca de que con las últimas versiones del kernel sí que irá bien, ¿qué hay de cierto en eso?

Respuesta

Ante todo quiero felicitarte por la decisión que has tomado. Poco a poco podrás comprobar como *Linux* es una plataforma ideal para realizar todo tipo de tareas, - desde la composición de documentos hasta la edición gráfica, pasando por el acceso a *Internet*.

Sin ningún lugar a dudas, la mayor crítica que se le puede hacer a *Linux* como sistema operativo es la relativa complejidad de su instalación. La mayoría de las personas que instalan *Linux* por primera vez encuentran alguna dificultad para instalar su *hardware* correctamente. En tu caso, el problema es el CD-ROM.

La gran mayoría, más del 99.99%, de los reproductores de CD que se instalan en los PC's utilizan un sistema de comunicaciones similar al de un disco duro. Para poder utilizarlos es necesario tener instalado soporte para ellos en el *kernel*. Cualquier distribución reciente de *Linux* incluye los *drivers* necesarios en la versión del núcleo que instala por defecto.

En caso de que no tengas los *drivers* instalados, tendrás que recompilar el *kernel* para incluirlos. Sin embargo, lo más probable es que no estés proporcionando a la orden *mount* los parámetros necesarios.

Para montar una unidad de CD IDE en tu sistema de ficheros, deberás realizar la siguiente secuencia de instrucciones:

- Lo primero que necesitamos es un punto de montaje, un directorio, vacío. Para ello, podemos crearlo o utilizar uno que ya exista en tu máquina. En la mía utilizo el directorio **/cdrom**. Si ya tienes un directorio creado, puedes pasar al punto siguiente:

```
mkdir /cdrom
```

- Una vez tenemos un punto de montaje, hemos de encontrar el fichero de dispositivo que corresponde a nuestra unidad de CD. Normalmente, la unidad se encuentra en **/dev/cdrom**, sin embargo es posible que el sistema de instalación no haya realizado el enlace correspondiente. En este caso, - suponiendo que tenemos un CD IDE, - buscaremos en **/dev/hd***, donde el valor que toma el asterisco depende de donde esté conectado el CD. Usaremos */dev/hdb* si el CD está conectado como esclavo del primario; */dev/hdc* si es maestro del secundario y */dev/hdd* si es esclavo del secundario. Normalmente, si sólo tenemos un disco duro en la máquina suele ser el maestro del secundario (*/dev/hdc*).

- Una vez sabemos donde está nuestra unidad, llamaremos a la orden *mount*. La sintaxis que vamos a utilizar es la siguiente:

```
mount -t iso9660 /dev/hdc /cdrom
```

Una vez ejecutado el comando, el sistema deberá montar el disco de la unidad en el directorio */cdrom*. Normalmente obtendremos el siguiente mensaje:

```
mount: block device /dev/hdc is write-
protected, mounting read-only
```

El cual nos indica que la operación ha tenido éxito, aunque el dispositivo se monta en modo de *sólo lectura*. Sin embargo, si la operación no ha tenido éxito, podemos obtener los siguientes mensajes:

```
mount: /dev/hdc already mounted or
/cdrom busy
mount: fs type iso9660 not supported by
kernel
mount: the kernel does not recognize
/dev/hdc as a block device
mount: wrong fs type, bad option, bad
superblock on /dev/cdrom,
or too many mounted file systems
```

```
mount: /dev/cdrom is not a valid block
device
```

El primero de ellos puede ser debido a dos cosas: el CD ya está montado en el sistema (la orden *mount* sin parámetros nos informa de los sistemas de ficheros montados) o bien hay alguien utilizando el directorio en cuestión.

El segundo se debe a que el sistema de ficheros del CD no está soportado por el *kernel* de nuestra máquina. Para poder utilizar discos de datos deberemos recompilar el núcleo añadiendo soporte para *ISO9660*.

El tercero nos informa de que no hay nada conectado a */dev/hdc*. En este caso deberemos probar con los demás dispositivos. Si ninguno funciona, deberemos sospechar que no hay soporte para nuestro lector de CD en el *kernel* actual, por lo que tendremos que recompilarlo añadiendo soporte para IDE/ATAPI CDROM.

El cuarto es debido a que hemos introducido un CD de música o a que el disco está defectuoso. Finalmente, el último mensaje se deberá probablemente a que no tenemos un CD de datos insertado en el lector.

En cualquier caso, si has instalado tu distribución de *Linux* a partir del CD-ROM, es prácticamente seguro que tu *kernel* lo soporta. Simplemente revisa la sintaxis de tu orden *mount* siguiendo las directrices indicadas.

Suele ser interesante, sobre todo a la hora de compilar aplicaciones distribuidas como código fuente, tener instaladas las fuentes del *kernel* en nuestra máquina. Sin embargo, para la mayoría del *hardware* que podemos encontrar en un PC no es necesario ir a buscar las últimas versiones.

Cualquier *kernel* 2.0.X será más que suficiente para soportar un CD-ROM IDE. Claro que si lo que quieres es sacar el máximo rendimiento a un sistema con dos procesadores, exportando discos por NFS y utilizando tarjetas especiales multipuerto accediendo a sistemas de ficheros exóticos,necesitarás una versión experimental de la serie 2.1.

En cuanto a instalar un nuevo *kernel* en el sistema, no hace falta tener todos los anteriores. Basta con tener las fuentes del que se quiera instalar. Supongamos que tienes una versión en tu CD; si consigues montar el CD, podrás acceder directamente a ella. En caso contrario te sugiero que la copies desde DOS o Windows95 en tu disco duro y luego la copies a la partición de *linux*.

El fichero *kernel* que tengas en el CD-ROM puede tener típicamente dos extensiones: *.rpm* y *.tgz*. El primero de los casos lo que requiere es que tengas instalada una versión de *RedHat Linux*, pudiendo instalarla en tu máquina con la orden:

```
rpm -i linux-2.0.35.rpm
```

El segundo de los casos es más general, ya que lo que tienes es un archivo *tar* comprimido con *gzip*. Para instalar esta versión deberás seguir los siguientes pasos:

- Colocarte en el directorio */usr/src* y copiar allí el *tgz* del *kernel*.
- Borrar el directorio *linux*, si es que existe. La orden para hacerlo es *\rm -Rf linux*.
- Descomprimir el *tgz* mediante la línea: *tar xvfz linux-2.0.35.tgz*.

En ambos casos te encontrarás un nuevo directorio llamado *linux* dentro de */usr/src*. Esta es la localización de tus recién instalados *fuentes* del *kernel*.

Para compilarlos de una manera cómoda, te aconsejo que lo hagas desde un *xterm* dentro del entorno gráfico de *X-Windows*. En cualquier caso, también se puede hacer desde la consola.

Los pasos que hay que dar son los siguientes:

1. Entramos en el directorio del *kernel*: *cd /usr/src/linux*.
2. Configuramos todos los parámetros, del modo siguiente:
   - Si estamos en la consola: *make menuconfig*.
   - Si hemos arrancado las *X-Windows*: *make xconfig*.
3. Una vez activado el programa de configuración, seguiremos las distintas opciones que se nos ofrecen hasta completar todos los campos. No olvides activar soporte para IDE/ATAPI CDROM y ISO9660 *filesystem*.
4. Salimos del programa guardando el fichero de configuración.
5. Ejecutamos la orden: *make dep ; make clean*. De esta manera preparamos el *kernel* para el proceso de compilación y limpiamos la basura que pudiera haber de procedimientos anteriores.
6. Arrancamos la compilación con *make zImage*.

Completados estos pasos, sólo queda esperar. La cantidad de tiempo depende de la máquina - en un AMD K6-233 suele tardar unos 4 minutos. Al finalizar la misma, el nuevo *kernel* se encuentra en:

```
/usr/src/linux/arch/i386/boot/
```

El nombre del fichero, de unos 500 *Kbytes* de tamaño será:

```
zImage
```

Una vez compilado el núcleo, deberás configurar tu sistema para que arranque a partir de él. El procedimiento depende del sistema que utilices para hacerlo, ya sea LILO o LoadLin.



HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS

DRUMBEAT 2.01

Generación, sin código, de elementos visuales para la web. Se trata de una solución especialmente indicada para la creación de tiendas virtuales y catálogos interactivos en Internet. Tiene la ventaja de ser una opción rápida y sencilla de conseguir resultados muy profesionales.

INSTABASE WEB-READY 4.0

Creación personalizada de bases de datos para la web. Los campos de las bases de datos pueden ser completamente personalizados e incluso en los registros pueden ser utilizadas imágenes. Dispone de soporte para registros ilimitados.

KAWA 3.10

Entorno integrado para el desarrollo en Java. Dispone, además del propio módulo de programación, otro que permite el aprendizaje del lenguaje Java. Proporciona además un interfaz visual que dispone de un gestor de proyectos con depurador, un gestor de clases, un gestor de código y un editor de clases.

CUTE-PACKAGER 1.6

Entorno muy flexible para la creación de paquetes comprimidos de archivos. Se trata de un asistente para la creación de ficheros comprimidos con extensión .CAB y .ZIP así como ejecutables autoextraibles .EXE. No precisa disponer de conocimientos e incluso puede proteger los ficheros con un password. Es Freeware.

NETOBJECTS BEANBUILDER 1.0 FINAL

Entorno para el desarrollo de aplicaciones sin necesidad de utilizar código fuente. Se puede utilizar para desarrollar aplicaciones Java, applets Java y JavaBeans.

NETOBJECTS FUSION 3.01

Suite profesional para la creación de sites web. Proporciona al Webmaster de toda la potencia que precisa para acometer proyectos de calidad. Es una de las aplicaciones más utilizadas profesionalmente. No es un simple editor de páginas HTML sino que incorpora opciones para implementar plantillas y mantener un site completo.

SUPERCEDE FOR JAVA 2.0

Potente entorno de desarrollo en Java. RAD que permite la creación de aplicaciones Java. Permite realizar un desarrollo siguiendo la norma, edición, compilado, carga y testeo del programa. Trabaja de modo interactivo ya que permite la modificación y la depuración en tiempo real, dispone de un interfaz que centraliza todas las funciones e incluye la posibilidad de reutilizar componentes. Puede generar tanto código Java como ejecutables nativos de Windows.

UNIXDOS TOOLKIT 4.2

Sistema operativo Unix desde la línea de comandos de DOS. Permite acceder a más de 90 utilidades a través de la línea de comandos del DOS. El paquete incluye 64 utilidades Unix, 9 utilidades Berkeley Unix, desarrollo de scripts, gestión de ficheros, etc.

LENGUAJES

VISUAL BASIC

CGI WORKERMAN 1.0

Procesamiento de scripts CGI con Visual Basic. Es un pequeño módulo cliente-servidor para la creación y gestión en Visual Basic de pequeños scripts CGI. Permite la codificar direcciones URL y el proceso de funciones simples. Muy interesante cuando se trata de introducir Visual Basic en servidores que no soportan CGI.

LEARN VISUAL BASIC 6 V1.0

Tutorial de gran calidad para el aprendizaje de la programación en Visual Basic. Permite desarrollar conceptos de la programación en este lenguaje y conocer las herramientas disponibles. Muestra las capacidades de diseños, implementación y distribución de las aplicaciones desarrolladas en Visual Basic. Combina más de 450 páginas con más de 60 ejemplos de aplicaciones. En esta versión sólo se incluyen las cuatro primeras lecciones.

JAVA

JSUITE 1.1

Colección de JavaBeans. Recopilación muy interesante de JavaScripts que

pueden ser utilizados libremente en cualquier Website. Incluyen desarrollos en Java de relojes, efectos de imagen, calendarios, links aleatorios, etc.

NETSCAPE VISUAL JAVASCRIPT 1.0

Creación de codigo JavaScript de manera rápida. Herramienta para el desarrollo de componentes Java de Netscape. Permite programar de manera visual y está especialmente indicada para los programas servidor de Netscape.

OBJECTSPACE VOYAGER 2.0

Herramienta de diseño Java de creación rápida de aplicaciones. Se trata de un avanzado ORB (Object Request Broker) especialmente diseñado para la distribución de desarrollos Java. Es 100% Java compatible y usa el lenguaje de modelado de objetos de éste para construir aplicaciones remotas, enviar mensajes y compartir información entre programas. Es Freeware.

HTML

DYNAMIC HTML EDITING COMPONENT SDK

Kit de desarrollo de HTML dinámico de Microsoft. Incluye un control ActiveX que puede ser usado para añadir características de edición a aplicaciones y páginas web independientemente del lenguaje que se utilice. Ejemplos y demos que muestran como se deben usar los controles y los componentes, incluyendo C++, Visual Basic y aplicaciones basadas en la web.

HOMESITE 4.0 BETA RC-3

Excelente editor de páginas HTML que incluye capacidades tan interesantes como gran variedad de plantilla prediseñadas, una ventana de visualización rápida, un asistente para la creación de frames, soporte para hojas de estilo en cascada y controles ActiveX, posibilidad de configuración del color de muestra de las etiquetas HTML, capacidad para múltiples deshacer, creación rápida de tablas y listas, verificación de links, soporte JavaScript etc.

INFOCOURIER 1.32 HTML COMPILER

Compilador de código HTML para su distribución en disquetes. Interesante herramienta que tiene como mayor capacidad, la posibilidad de portabilizar los sites web. Para ello realiza una compresión interna y permite que las páginas web sean navegables. Especial para distribución en disquetes.

PRETTY HTML 1.0

Excelente utilidad para el formateo y compresión de código. Herramienta que ayuda a los desarrolladores de código HTML a estructurar su información y comprimir al máximo sus desarrollos. De una manera muy sencilla formatear el código en colores para una más sencilla localización. Tiene la ventaja de que ocupa muy poco espacio y precisa de mínimos requerimientos para su utilización.

WEBCOMPILER 98 1.0

Compilador de páginas HTML para la creación de ejecutables navegables. Curiosa utilidad que permite la conversión de un sitio web en un simple fichero ejecutable autoextraible que posibilita su posterior navegación. Su interfaz trabaja a modo de navegador.

OTROS

ADVANCED MENUS 1.01

Colección de componentes para la gestión de menús en Delphi. Recopilación de componentes para la gestión de menús en Delphi Incluye además un elemento para el acceso a cada uno de los items de menú del sistema, otro para el acceso total a los menús del sistema y un tercero para unir un menú estándar con un popup. Es Freeware e incluye los todos los códigos fuente.

INF-TOOL 3.85

Creación de procesos de instalación para aplicaciones. Permite la creación de procesos de instalación a través de ficheros .INF soportados por Windows 95, 98 y NT. Tienen la ventaja de que pueden ser modificados con tan sólo editar el fichero de texto y su ocupación en disco es mínima. Esta utilidad permite tomar un control total sobre este tipo de ficheros.

DELPHI VCL EXTENSIONS (RXLIB) 2.50

Colección de 50 componentes Delphi. Paquete que incluye variados objetos y rutinas para Delphi y C++ Builder. Incluye componentes nativos, objetos para trabajar con bases de datos, manipulación de rejillas, edición de fecha, formularios, SQL, importación de texto rico (RTF), etc.

SYLVIE 1.36D

Curioso desarrollo de un robot animado que mantiene conversaciones. Se trata de un personaje virtual que dispone de personalidad artificial. Tiene habilidades para comprender y hablar en inglés gracias a una tecnología de lenguaje natural. Basta con teclear frases con el teclado y Sylvie responde inmediatamente. Puede además leer archivos de texto.

TVICHW32 2.0

Rutinas para el acceso a recursos de 32 bits. Ofrece a los desarrolladores un acceso directo al hardware usando el Windows DDK. Provee un soporte transparente a las funciones 32 bits de Windows proporcionando métodos estándar para el acceso directo en tiempo real al puerto I/O, memoria I/O y a las interrupciones hardware sin necesidad de escribir drivers virtuales. Incluye las librerías DLL para Visual C++, los OCX para Visual Basic y los VCL para Delphi y C++ Builder.

VBOX BUILDER 4.1

Creación automática de versiones de demostración. Es una utilidad libre de royaltys que se muestra como una solución que permite a los desarrolladores la creación software de tipo trial sin modificar el código fuente. Las trials pueden estar a tiempo de uso, número de ejecuciones, etc. Especialmente indicado para la venta de software a través de Internet.

W32DASM 8.5

Debugger muy versátil especial para desarrollos Windows. Es un desensamblador y debugger que incluye interesantes funciones de búsqueda y ejecución de pruebas. Incluye además una tabla de referencias cruzadas para llamar a instrucciones y soporte para MMX. Interpreta automáticamente funciones KERNEL32, USER32, SHELL32, COMDLG32, COMCTL32, GDI32, y ADVAPI32 API.

HERRAMIENTAS

CODEDIGGER 2.11

Utilidad de búsqueda de texto y código en archivos de texto. CodeDigger ofrece una forma flexible de realizar búsquedas en código bajo parámetros. Es una utilidad muy rápida y dispone de un diseño multibúsqueda. Los archivos elegidos dentro del criterio de búsqueda son mostrados con la sintaxis del código coincidente resaltada en un color distinto. Soporta los más populares lenguajes de programación

EDITPAD 3.3.3

El perfecto sustituto para el Bloc de Notas de Windows. Editor de textos muy potente y que precisa de muy pocos requerimientos. Trabaja bajo un interfaz multiventana que se gestiona a través de pestañas. Muy rápido.

GO!ZILLA 3.1

Impresionante gestor de "downloads" para automatizar procesos y solucionar problemas. El programa permite continuar la bajada de un fichero tras el corte de la conexión. Para comenzar el download basta con arrastrar el link desde el navegador a Go!Zilla. Muestra información sobre el archivo como su tamaño o el tiempo estimado de descarga.

NORTON ZIP RESCUE 1.0

Recuperación de datos a través de unidades Zip de Iomega. El programa crea un disco Zip de emergencia que contiene una versión reducidad de Windows, el software Norton Rescue y la configuración del escritorio. Esto permite una sencilla recuperación de muchos de los problemas más típicos de Windows, incluidos los desperfectos en el registro del sistema, tablas de partición dañadas, contaminación de virus, perdidas de datos o daños en el sector de arranque.

POINTCAST NETWORK 2.6

Cliente que permite la recepción de información personalizada de Internet. Permite recibir, al minuto, los datos requeridos sobre noticias, cotizaciones bursátiles, información meteorológica, deportes, informática, y mucho más. Sólo recibes lo que quieres en el momento que quieres. Basta con seleccionar los canales de tu interés y pulsar sobre un botón para recibir la información. Luego la información puede ser consultada sin necesidad de conexión a Internet. Dispones, entre otros, de canales tan importantes como la CNN. Su uso es libre y gratuito.

* WINZIP 7.0 VERSIÓN FINAL

La última versión del estándar de compresión. Se ha convertido en una herramienta casi imprescindible en todo PC. Es el estándar para la gestión de archivos comprimidos, sea cual sea el formato. Gracias a su potencia, permite realizar operaciones avanzadas.

REDES

LOCALES

FICTIONAL TELNET DAEMON 2.1

Servidor Telnet que trabaja como Demonio. Es un pequeño programa que espera la acción del programa cliente y procesa todas las actividades de envío de información que realiza éste.

ARTISOFT LANTASTIC 8.0

Entorno muy potente para gestión para redes locales. Permite, bajo Windows 95 y Windows 98, la compartición de archivos, aplicaciones, impresoras y unidades de disco. Especialmente indicado para redes locales montadas en Pymes ya que facilita mucho el trabajo a los usuarios con menos conocimientos informáticos.

VTREX TELNET DAEMON 2.1

Demonio Telnet. Es un simple Telnet que funciona como un demonio bajo Windows NT. Se ejecuta como un servicio más de NT y es compatible con la gran mayoría de aplicaciones de línea de comandos Telnet. Se incluye el código fuente en C++ Builder.

DISTRIBUIDAS

LANTRACE 3.0

Analizador de protocolos y tráfico de Internet. Especialmente indicado para redes con proxy con varios accesos a Internet. Permite visualizar todos los movimientos y conexiones de la red y ofrece estadísticas sobre los paquetes de datos que se desplazan.

* WINGATE 2.1D

Compartición de módems a través de una LAN. Permite conectarse a Internet a través de varios PCs con un única línea telefónica. Además permite realizar cualquier operación como si se tuviera el módem en el equipo local.

WHITE PAPER GROUP POLICY NT 5.0

Información especial sobre la administración del futuro NT 5.0. Información muy interesante para los usuarios, ofrecida en exclusiva por Microsoft, de características técnicas sobre la administración de redes locales en el futuro Windows NT 5.0.

 OTROS

ARTISOFT XTRAMAIL 1.11

Excelente servidor de correo electrónico para Pymes. Especialmente diseñado para la gestión del correo electrónico en pequeñas empresas. Muy robusto y escalable posibilita la asignación de direcciones individuales para cada uno de los usuarios de la compañía. Gestiona tanto el envío como la recepción de los mensajes y los posiciona en los buzones adecuados.

ARROW MAILING LIST SERVER 1.3

Creación y gestión de listas de correo usando una conexión E-mail estándar. Permite una creación y administración avanzada de listas de correo. Para ello basta con disponer de una simple cuenta de correo electrónico. Facilita la moderación de los foros automatizando procesos de respuesta.

COFFEECUP DIRECTFTP 2.0

Cliente FTP con características especiales para web publishers. Es un interesante programa que dispone de características como drag and drop, previos de imágenes y edición de código HTML directamente en el servidor remoto.

INDEXSITE 1.0

Herramienta servidor de indexación de información web. La idea de este programa es crear la estructura de directorios de un site para volcarla en una base de datos. Es muy interesante para disponer de una presentación visual de la situación de cada una de las páginas y la información de cada una de ellas.

NEOTRACE 1.21

Traceador gráfico de alta velocidad para detectar problemas de conexión. Se trata de un programa muy sencillo y extremadamente rápido para devolver la información. Es muy interesante para detectar problemas en la red y para obtener información de Internet. También puede ser muy útil para "cazadores de spam".

WHITE PAPER MICROSOFT WINDOWS NT 5.0 BACKGROUNDER

Información especial sobre características técnicas del futuro NT 5.0. Documentación de gran interés, ofrecida en exclusiva por Microsoft, de características técnicas sobre el futuro Windows NT 5.0.

 DOCUMENTACIÓN Y TUTORIALES

ASCII CATALOG 5.0

Guía de referencia para programadores en general. Es una excelente colección de mapas de caracteres, tablas de conversión y guías de referencia especialmente indicada para desarrolladores, diseñadores HTML, articulistas y diseñadores gráficos.

JAVA-GODFATHER 1.1B

Completa documentación sobre programación de Java y Java Beans. Todo lo necesario para aprender a programar en Java. Se trata de un curso muy completo, en formato ayuda de Windows, que permite solucionar dudas sobre el lenguaje del futuro.

MICROSOFT HTML HELP WORKSHOP 1.1

Entorno para la creación de ficheros de ayuda Windows. Es un programa que permite al usuario la creación personalizada de archivos con extensión .HLP para su distribución con aplicaciones recién creadas. Está compuesto por una utilidad para gestionar el diseño del proyecto, un compilador y un editor de imágenes.

LEARN VISUAL BASIC 6 V1.0

Tutorial de aprendizaje para la programación Visual Basic. Permite desarrollar conceptos de la programación en este lenguaje y conocer las herramienta disponibles. Muestra las capacidades de diseños, implementación y distribución de las aplicaciones desarrolladas en Visual Basic. Combina más de 450 páginas (en formato MS Word) con más de 60 ejemplos.

VBA TUTOR 1.0

Tutorial de aprendizaje de Visual Basic para Aplicaciones. Documenta cómo se debe usar Visual Basic para Aplicaciones para sacarr el mayor partido y personalizar a nuestro antojo Office 97. Consta de 10 tutoriales y tres amplios ejemplos de desarrollos. Se encuentra en documentos de texto con formato Word. También se incluyen algunas macros prediseñadas.

FUENTES

Atículo de portada: HTML Dinámico y las capas (I)
Un pequeño ejemplo de la correcta utilización de las capas en HTML Dinámico.

Reconocimiento de Voz (IV)
Últimos pasos, todo lo necesario para poder crear una buena aplicación.